深入浅出 **Android**

1 入门 4

初探 ANDROID 4

2008 年末最大的冲击 4

Android 是什么 4

从创意开始 5

安装 ANDROID 开发工具 7

InstallAndroid 7

安装流程 8

开启现有工程 11

开启工程 11

导入工程 11

修复工程 11

操作 ANDROID 模拟器 12

使用 Android 模拟器 12

列出模拟器类型 12

建立模拟器 13

列出已建立的模拟器 14

移除模拟器场景 16

移除模拟器 17

建立一个 ANDROID 程序 18

建立新工程 18

初识 GUI 界面 21

描述使用者界面 21

设计 GUI 界面 25

视图(View) 25

查阅文件 25

离线文件 25

视觉化的界面开发工具 28

获取标识 ID 29

存取识别符号 29

新增 XML 文件 33

解读程序流程 35

解读程序流程 35

完成 BMI 程序 39

完成 BMI 程序 39

初级 43

重构程序 43

什么是重构 43

MVC 43

初识 INTEN 53

初识 Intent 53

加入菜单 57
加入菜单(Menu) 57
定义 ANDROID 清单 60
加入新的 ACTIVITY 64
独立的 Activity 64
传送数据到新的 ACTIVITY 69
传送数据到新 Activity 69
活动的生命周期 77
生命周期 77
Active（活动） 78
Paused（暂停） 78
Stopped（停止） 78
Dead（已回收或未启动） 78
存储信息 83
发布到市集(MARKET) 87
Android Market 87
***Android Market* 的运作方式** 87

深入浅出 Android

1 入门

初探 Android

2008 年末最大的冲击

目前有 15 亿使用者可以透过电脑，在网路上看到 Google 广告。
若手机使用者也能上网，其潜在使用者则可以达到 30 亿。

2007 年 11 月 5 日，Google 与其他 33 家手机制造商(包含摩托罗拉、宏达电、三星、LG)、手机晶片供应商、软硬件供应商、电信业者所联合组成的开放手持装置联盟(Open Handset Alliance)，发布了名为「Android」 的开放手机软硬件平台。参与开放手持装置联盟的这些厂商，都会基于 Android 平台，来开发新的手机业务。

紧接着，在 Android 平台公布的一周之后(11 月 12 日)，Google 随即发布了可以免费自由下载，能在 Windows、Mac OS X、Linux 多平台上使用的 Android 软件开发工具(Software Development Kit, SDK) 与相关文件。间隔数天，又再次发布操作系统核心(kernel)，与部分驱动程序的源代码。 一项展示 Google 欲将手机这个现代人的随身工具推往开放平台， 让人们可以自由修改创作出符合自己需求的手机应用的决心。

身为 Google 公司创办人之一的 Sergey Brin，也在 Android 软件开发工具(SDK)发布的同时，现身于视讯广告影片中，为大众介绍 Android 平台。 Sergey Brin 也同时宣佈举办总奖金高达 1000 万美元(3 亿多台币)的开发者大奖赛， 鼓励程序开发者去深入探究 Android 平台的能力。写出具创意、实用性十足、能提供使用者更好的手机使用经验的应用程序。

2008 年 9 月 24 日，T- Mobile 首度公布第一台 Android 手机(G1)的细节，同日 Google 也释出了 Andorid SDK 1.0 rc1。对应用程序开发者而言，1.0 代表了开发者可以放心地使用 API，而不必再担心 API 有太大的变动。 G1 在同年 10 月 20 正式发售。在发售前仅针对原 T-Mobile 用户的预购活动中，已经被预购了 150 万台。在 10/21 日，Open Handset Alliance 公开了全部 Android 的源代码。从此，开发者拥有了一个完全开放的手机平台。

Android 是什么

在可见的将来，基于 Android 平台的手机程序设计，将像今日的 PC 程序设计一样普及。

「Android」是一个基于 Linux 核心(kernel) 的开放手机平台操作系统。与 Windows Mobile、Symbian 等手机操作系统处在同一级别。

对于设备制造商来说，「Android」是一个免费的平台。「Android」操作系统让设备制造商免除「每出一台手机，就得被手机操作系统厂商(如 MicroSoft)收取费用」的情况。对硬体开发厂商来说，「Android」也是个开放的平台。只要厂商有能力，可以在这个平台上自由加入特有的装置或功能，不受手机操作系统厂商的限制。

对于手持装置的开发者来说，「Android」是个先进的平台。平台上的应用程序可相容于各种型号的 Android 手机，免去为各种不同手机机型开发的困扰。「Android」平台支援各种先进的网路、绘图、3D 处理能力，可以用来提供更好的使用者体验。

对于使用者来说，「Android」是一个用于手机的操作系统。使用者只要先申请一个免费的 Google 帐户，当使用者想换一台手机时，就可以在不同厂牌，同样使用「Android」操作系统平台的手机之间选择，并且很容易地将如联络簿等个人资料转换到新手机上。

Android 在 Linux 核心的基础上，提供了各种合用的函式库，和一个完整的应用程序框架。并采用较符合商用限制的 Apache 版权。在 Linux 核心的基础上提供 Google 自制的应用程序运行环境 (称作 Dalvik，与 Sun 的 J2ME 不同)，并提供基于 Eclipse 整合开发环境(IDE)的免费、跨平台(Windows、Mac OS X、Linux)开发工具(SDK)，便于应用程序开发者学习、使用。

免费、熟悉的跨平台开发工具，让具备一些物件导向观念，或视窗程序开发经验的开发者，能在一定时间内上手。1000 万美元的大奖赛则提供了足够的诱因，让第一支 Android 手机正式面市前，就拥有了各式各样的应用程序可供使用。

从不同角度来说，Android 代表着

- 一个崭新的开放源代码操作系统平台、
- 一个友善的免费应用程序开发环境、
- 一个与世界各地的程序开发者，站在相同起跑点上的公平竞争机会。

从创意开始

预测未来的最好的方式，就是去创造未来

你是否曾经思考过，当我们拥有了一台能够上网，能够依照自己的期望自由修改、调试功能的手机，我们会用它来施展什么创意？Android 大奖赛首页上提供了一些方向。

你的心里可能已经有了关于手机程序的绝好创意，你想在 Android 开放手机平台上，实现因为其他封闭平台的种种限制，而无法在手机上达成的需求；无论你以何种目的来看待「Android」平台，都需要为了达成你的创意， 所需的一些基本协助。 本书所提供的内容，将可以协助你快速地理解 Android 平台的概念。 提供你初次开发手机程序界面的注意事项、Android 资料应用程序的运作过程， 与详细的 Android 应用程序实例解说。

要开发 Android 应用程序，你甚至不需要拥有实机。Google 已经为我们提供了各个主要平台(Windows、Mac、Linux) 上可用的开发工具包。开发工具包中，也包含了 Android 手机模拟器，好让我们在电脑上就能完成所有的手机应用程序开发工作。接着，我们就先来安装 Android 开发工具吧。

参考资料

* Android 官方网站 http://www.android.com/
* 开放手持装置联盟(Open Handset Alliance) http://www.openhandsetalliance.com/
* Google Android 开发者部落格 http://android-developers.blogspot.com/
* Android 开发者大赛网站 http://code.google.com/android/adc.html
* Android 文件 http://developer.android.com/
* http://www.onlamp.com/pub/a/onlamp/2007 ... e-sdk.html
* CNet 专访：Google 手机平台背后的原创者
http://www.zdnet.com.tw/news/comm/0,200 ... 898,00.htm
* Android 源代码网站 http://source.android.com

安装 Android 开发工具

InstallAndroid

安装 Android 开发工具

Android 提供免费而且跨平台的整合开发环境，只要电脑能连接上网路，我们随时都能下载相关工具下来，并开始开发 Android 应用程序。 有了轻便易用的开发工具，我们可以把心力专注于如何将想法实现到应用程序上。

系统需求

撰写 Android 的应用程序，需要一套个人电脑系统。至于操作系统的部份，几个主流操作系统都有支援。

支援的操作系统如下：
* Windows XP 或 Vista
* Mac OS X 10.4.8 或之后版本 (适用 x86 架构的 Intel Mac)
* Linux (官方于 Ubuntu 6.10 Dapper Drake 上测试)

我们需要安装一些 Android 开发环境所需的程序工具，这些工具都是可以免费上网取得的：

* JDK 5 或 JDK 6

你需要安装 Java 开发工具 (JDK 5 或 JDK 6)。 只安装 Java 运行环境(JRE) 是不够的，你需要安装 Java 开发环境 (JDK)。

你可以在命令行上输入「java -version」 来查看目前系统上已安装的 java 版本(java 版本需 >1.5)。

要注意的是 Android 与 Java Gnu 编译器 (gcj) 还不相容。

* Eclipse IDE，一个多用途的开发工具平台。

你可以下载安装 Eclipse 3.3 (代号 Europa) 或 3.4 (代号 Ganymede) 版。

请注意你选择的版本需包含 Eclipse Java 开发工具扩充套件 (Java Development Tool Plugin, JDT)。

大多数 Eclipse IDE 包中都已含有 JDT 扩充套件。若对 Eclipse 平台不熟悉的话，建议直接选择 「for Java Developers」版本来下载。

* ADT，基于 Eclipse 的 Android 开发工具扩充套件 (Android Development Tools plugin)。
* Android SDK，Android 程序开发套件，包含 Android 手机模拟器(Emulator)。
* 其他开发环境工具（非必要安装）

o Linux 和 Mac 环境上需要自动编译的话可以自行安装 Apache Ant 1.6.5 或之后版本，Windows 环境上则需要 Apache Ant 1.7 或之后版本。

o NetBeans、IDEA 等开发平台亦有推出自己的 Android 开发工具，但本书中还是以讨论官方基于 Eclipse 平台的开发工具为准，其他平台不予涉及。

安装流程

假设读者已先安装了 JDK 5 或 JDK 6。 那麼 Android 的安装流程可以分为以下五个步骤

1. 下载 Eclipse
2. 安装 Eclipse
3. 安装 ADT 扩充套件
4. 下载 Android SDK
5. 设定 Android SDK

详细的安装流程如下：

1. 下载 Eclipse

首先我们需要下载 Android 开发时会用到的整合开发环境 Eclipse。

目前 Android 应用程序只支援使用「Java」程序语言来编写 Android 应用程序。所以开发前必须先安装 Java 开发套件(Java Development Kit, JDK)。 各平台的 JDK 可至 http://java.sun.com 下载。

Mac OS X 操作系统则已内建 JDK。 安装好 JDK 后，我们可以前往 Eclipse 网站下载 Eclipse 这个方便的整合开发环境。

下载 Eclipse 时请选「Eclipse IDE for Java Developers」或「Eclipse IDE for Java EE Developers」这两种版本，只有这两种版本才会预装 Eclipse JDT 扩充套件。 范例中所选择的是「Eclipse IDE for Java Developers」版本。下载完同样先解压缩到适当目录下。

2. 安装 Eclipse

Eclipse 不需要安装，只要确认你的系统上有安装 Java，即可直接开启 Eclipse 文件夹，点击 Eclipse 开始执行 Eclipse 整合开发环境。

第一次启动 Eclipse 时会弹出视窗让你决定预设的工作目录。一般使用 Eclipse 预设的工作目录即可。进入到 Eclipse IDE 后，不必急着四处观望。我们先来安装 Android 开发工具扩充套件。

3. 安装 ADT 扩充套件

我们将在 Eclipse 上 安装 Android 开发工具 (ADT)。

Eclipse 3.4 版

找到屏幕上方的选单列， 选择 「Help->Software Updates」 选项，这选项会带出一个新视窗。

选择「Available Software」标题，选择右方的 「Add Site...」 (新增网站)按钮，会弹出一个输入框。

在输入框中的"Location"栏位中输入网址(URL)：

「https://dl-ssl.google.com/android/eclipse/site.xml 」

按下 "OK" 按钮离开。Eclipse 会花一点时间寻找合适的版本。如果不行的话可以尝试使用 http://dl-ssl.google.com/android/eclipse/, 有时候 https 是无法访问的。

在视窗中全选「 https://dl-ssl.google.com/android/eclipse/site.xml 」项目「Developer Tools」中的的选项后，按下右方的「Install」按钮。

按下 「Next」 (下一步)键。照着步骤就安装完成。安装完会提示需重新启动 Eclipse，按下 「Yes」 重新启动。

Eclipse 3.3 版

找到萤幕上方的选单列， 选择 「Help->Software Updates->Find and Install」 选项，这选项会带出一个新视窗。

选择 「Search for new features to install」 (搜寻新功能供安装)选项，按下 「Next」 (下一步)键。出现设定画面。

选择右上角的「New Remote Site」(新增远端网站)按钮，会弹出一个「New Update Site」(新增更新网站)输入框。

在输入框中输入扩充套件的名称(Name) 「ADT」 跟网址(URL) 「http://dl-ssl.google.com/android/eclipse/site.xml 」，按下 「OK」 按钮离开。

按下 「Finish」 按纽继续下一步。Eclipse 会花一点时间寻找合适的版本。

接着我们要做的，就是等 Eclipse 显示出选项时，勾选合适的版本安装。

安装完会提示需重新启动 Eclipse，按下 「OK」 重新启动。

离线安装

已经安装成功的读者可以跳过这段。有些读者因为网路环境的关係，无法顺利地直接线上安装 Android 开发工具。

这时我们可以先前往 http://developer.android.com/sdk/adt_download.html ，手动下载最新的开发工具版本来离线安装。

下载完最新的 ADT 扩充套件后，打开 Eclipse 开发环境，找到萤幕上方的选单列， 选择 「Help->Software Updates」 选项，

这选项会带出一个新视窗。选择「Available Software」标籤，选择右方的 「Add Site...」(新增网站)按钮，会弹出一个输入框。

选择右上角的「Local...」按钮，并选取刚下载的 Android 最新开发工具档桉，选到之后按下 "OK" 按钮离开。

在视窗中全选新出现项目的所有选项后，按下右方的「Install」按钮。Eclipse 会花一点时间开始安装 ADT 扩充套件。

4. 下载 Android SDK

接着我们要从 http://developer.android.com/ Android 官方网站下载 「Android 软件开发套件」 (Software Development Kit, SDK)。 下载下来的 SDK 档桉需要先解压缩。Windows 平台需要先另行安装解压缩程序，如免费的 7-zip 解压缩工具。

解压缩后会出现一个资料夹。为了之后描述方便，我们将解压缩后的 Android SDK 档桉夹命名为「android_sdk」。

5. 设定 Android SDK

打开偏好设定页面(Preference)，选择 Android 标签(请确认您已安装好 ADT 扩充套件，Android 标志才会出现在偏好设定页面中)，在 SDK Location 栏位按下 "Browse..."键，选择刚刚解压缩完的「android_sdk」档桉夹所在地，然后按下视窗右下角的套用(Apply) 按钮。这样一来，Android SDK 就算是设定好啦。

注解：若您安装过 SDK 1.5 版之前的版本，请先移除后再重新安装一次 ADT 扩充套件，才能顺利设定新版的 Android SDK。

方法是在萤幕上方的选单列，选择「Help > Software Updates」选项，在弹出的视窗上方点选「Installed Software」页面，选择「Android」开头的选项，点选右侧的「Uninstall..」按钮移除这些相关的插件。

下一步

设定好 Android SDK 后，我们就拥有了一个完整的 Android 开发环境。 我们先来看看 Android SDK 中提供的一些范例，

好了解 Android 到底能做些什麼。

参考资料

Eclipse 网站 http://www.eclipse.org/downloads/

安装扩充套件 http://developer.android.com/guide/deve ... s/adt.html

开启现有工程

开启工程

我们回到 Eclipse 环境来。在屏幕上方的选单列上，选择「File->New->Project」，会弹出「New Project」对话视窗。 Eclipse 是通用的编辑环境，可根据你所安装的不同扩充套件而支援许多种类的工程。点击「Android」资料夹下的「Android Project」，会开启「New Android Project」对话视窗。

我们将 开启 Android SDK 中提供的 ApiDemos 范例。在「New Android Project」对话视窗中，点选 "Browse..."按钮以选择「开启已经存在的工程」(Create project from existing source)。我们在此选择位于「android_sdk/platforms/android-1.5/samples」目录中的 Android 应用程序工程 (android_sdk/platforms/android-1.5/samples/ApiDemos)。

当我们选择了现存的范例程序工程时，「New Android Project」对话视窗中的诸如工程名称（Project Name）与属性等内容都将被自动填写好。这时我们可以按下 「Finish」按钮，完成从现存工程新增工程到 Eclipse 环境的动作。

导入工程

如果你的程序工程已位于工作环境(WorkSpace)资料夹下，想使用上述方法开启工程时，会得到欲开启的资料夹已在工作目录下的警告。因此我们得用另一个方法：导入工程。

在 屏幕上方的选单列上，选择「File->Import」选项，会跳出「Import」视窗。选择「General->Existing Projects into WorkSpace」项目，然后按下「Next」按钮带到新一个画面。在「Select Root Directory」栏位旁，按下右方的「Browse...」按钮，选择对应的工程。选择好后，按下「Finish」按钮完成从现存在工作环境 (WorkSpace)资料夹下的工程汇入到 Eclipse 环境的动作。

修复工程

完成新增程序工程到 Eclipse 后，我们可以在左侧的「Package Explorer」中找到我们新增的工程。

如果发现开启后的资料夹图示上有个小小的黄色惊叹号，表示这个工程汇入后还有些问题，我们可以使用 ADT 内建的功能来试着修复。 在「Package Explorer」的 「ApiDemos」 工程档桉夹图示上点选右键， 从「Android Tools」选单中选择「修复工程属性」(Fix Project Properties)。 (Android Tools->Fix Project Properties)

参考资料

* 如何开启 Hello World 程序 http://developer.android.com/guide/tuto ... world.html

操作 Android 模拟器

使用 Android 模拟器

我们已经透过「Eclipse」开发环境，开启了「ApiDemos」源代码。本章将讲解如何设定和操作 Android 模拟器。

设定 Android 模拟器

现在我们还不忙着开始写程序，先花点时间，来看看怎么在开发环境中，通过「Android 模拟器」来执行应用程序吧。

「Android 软件开发套件」（SDK） 1.5 以上的版本提供了支援不同版本模拟器的功能，在使用模拟器之前，必须先建立一个模拟器后才可在 Eclipse 开发环境中使用。

SDK 中提供了一个「android」命令行工具（在 android-sdk/tools 中），可以用来建立新专桉或是管理模拟器。在此我们使用「android」命令行工具来新建立一个模拟器。

在新的 Android 文件中，一律把 Android 模拟器称作「Android 虚拟机器」（Android Virtual Device），简写为「AVD」以作区别。

列出模拟器类型

首先，把「android-sdk/tools 」目录加入系统路径，我们以后就可以在任何地方使用「android-sdk/tools 」目录下的各种命令。

在 Windows 2000， XP， 2003 这些操作系统里，点选「我的电脑右键 > 属性 > 高级 > 环境变数」。在「系统变数(S)」栏中，选取「PATH」变数名称后，再点选「编辑(I)」按钮。

再此假设您安装 Android SDK 的路径是「C:\android-sdk\tools」，接着在弹出的视窗中将「;C:\android-sdk\tools」（注意要以分号隔开）这字串添在原本的字串之后，按下确定后重新启动操作系统。

重开系统后选择「开始 > 执行」，在弹出的输入框中输入「cmd」，即可开启命令行工具并继续以下的动作。

或是您也可以直接打开命令行，进入「android-sdk/tools 」目录，输入以下命令：

代码:

```
$ android list targets
```

在没有将 Android SDK 加入路径的情况下，在 Linux 或 Mac 环境中要输入

代码:

```
$ ./android list targets
```

萤幕上会列出所有支援的模拟器类型

代码：

```
$ android list targets
Available Android targets:
id: 1
    Name: Android 1.1
    Type: Platform
    API level: 2
    Skins: HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P
id: 2
    Name: Android 1.5
    Type: Platform
    API level: 3
    Skins: HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P
id: 3
    Name: Google APIs
    Type: Add-On
    Vendor: Google Inc.
     Description: Android + Google APIs
      Based on Android 1.5 (API level 3)
      Libraries:
            com.google.android.maps (maps.jar)
           API for Google Maps
            Skins: HVGA (default), HVGA-L, QVGA-P, HVGA-P, QVGA-L
```

在这边列出了三种模拟器类型。分别是编号(id)为 1、2 的 Android 1.1、1.5 模拟器，与编号(id)为 3 的「Google APIs」，Google 把自己提供的应用程序（如 Google Map）放在「Google APIs」这个模拟器类型中，因此要开发 Google Map 等 Google 专属应用程序时，就必须先建立编号 3 这类型的模拟器，稍后才能在适当的模拟器上作验证。

建立模拟器

我们现在来建立一个基本的 Android SDK 1.5 模拟器。

在命令行中输入以下命令：

代码：

```
$ android create avd --target 2 --name cupcake
```

这段命令的意思是：使用「android create avd」命令来建立一个新的模拟器，「 --target 2」参数的意思是这个模拟器使用 id 为 2 的模拟器类型（Android 1.5），「--name cupcake」参数的意思是将这个建立的模拟器命名为「cupcake」。

产生的结果如下

代码：

```
$ android create avd --target 2 --name cupcake
Android 1.5 is a basic Android platform.
Do you wish to create a custom hardware profile [no]
Created AVD 'cupcake' based on Android 1.5
```

列出已建立的模拟器

我们可以使用 「Android」命令行工具提供的「list avd」命令，来列出所有我们已经建立的模拟器。

在命令行中输入以下命令：

代码：

```
$ android list avd
```

产生的结果如下：

代码：

```
$ android list avd
Available Android Virtual Devices:
    Name: cupcake
    Path: /Users/mac/.android/avd/cupcake.avd
    Target: Android 1.5 (API level 3)
   Skin: HVGA
```

使用「 android list avd」命令看到有输出，即表示已成功建立模拟器，可以回到 Eclipse 环境来，设定执行应用程序所需的环境参数了。

设定环境参数

要执行 ApiDemos 程序前，我们得在开发环境中，事先设定好一些用来执行 ApiDemos 程序的环境参数。 以后使用其他程序专桉时，我们也能用同样的方式，让这些程序在我们的开发环境中运行。

首先，我们透过选单列上的「Run」(执行)选单，选择「开启执行参数设定」(Run-> Debug Configurations...) 进入运行环境参数设定画面。

进入设定画面后，在视窗左侧会有一整排 Eclipse 支援的运行设定，我们从中找到 "Android Application"(Android 应用程序)项目，按下滑鼠右键，点选 "New"(新增)选项。

选择 「New」 选项后，在「Android Application」项目下方会多出一列执行项目。

我们可以在 Name 栏位上输入一个代表这个环境参数的名称，在此我们输入与专桉名称相同的「ApiDemos」。

在「Project」栏位右方，点选「Browse...」按钮，开启「选择」（Project Selection）视窗，选择「ApiDemos」专桉并点选「OK」按钮，以选择要执行的工程。

在 「Launch Action」 选单中，确认预设选择的是「Launch Default Activity」。

至此我们就完成了模拟器环境参数的设定。 点选右下角的「Debug」按钮，Eclipse 就会启动 Android 模拟器。

小技巧:

在选单列中，也可以选择设定「Run Configuration...」选项。这时我们得到的是一个几乎完全相同的环境参数设定画面，只是右下角的「Debug」按钮变成了「Run」按钮。「Debug」与「Run」模式的环境参数设定可以共用，差别在于「Debug」模式下可以使用在之后章节中会介绍的 logd，来显示一些开发时所需的额外讯息。

再次启动 Android 模拟器

当我们设定好之后，以后碰到要再次启动模拟器的情况时，只要在萤幕左上角的「Debug」或「Run」图示右侧小箭头上按一下，从弹出的选单中选择刚刚设定的环境参数名称，模拟器即开始执行，并安装好我们所指定的专桉应用程序。

操作 Android 模拟器

切换模拟器场景

在命令列上执行「android list targets」命令后，我们可以看到萤幕上列出所有支援的模拟器类型。举我们刚才建立过的第二种类型（id 2）模拟器为例，列出讯息如下：

代码:

```
id: 2
    Name: Android 1.5
    Type: Platform
    API level: 3
    Skins: G1, HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P
```

其中 Skins 栏位中会列出所有支援的模拟器场景。

预设有「HVGA」与「QVGA」两种画面配置选项可选择，「HVGA」与「QVGA」又可以再各自分为「-L」(landscape, 横式) 与「-P」(portrait 直式)。

要建立「QVGA」模式的模拟器，则在前一节「android create avd」命令后，附加上「--skin QVGA」选项即可。要将预设的「HVGA 直式」显示改为横式，则可以透过使用快速键，直接切换萤幕来达成。

切换萤幕

在 Windows 操作系统上按下 「Ctrl」和「F12」键 ，或是在 Mac OS X 操作系统上同时按下「fn」 和「7」键，萤幕就会从预设的直式显示改成横式显示，再按一次则切换回原来的直式显示。

新增模拟器外观设定

Android 模拟器的用途，就是协助我们在电脑上也能模拟真实手机的动作。不禁会想，如果模拟器除了所模拟的动作之外，模拟器的外观也能跟实机长的一样，那不是整个更拟真，开发起来更有感觉吗？

没错，所以 Android 模拟器也允许使用者自行製作模拟器外观！

除了预设的模拟器外观之外，以世界第一台发售的 Android 手机「T-Mobile G1」为例，有人已经作好了「T-Mobile G1」的模拟器外观。可以前往 http://www.jsharkey.org/downloads/G1.zip 下载。

要新增模拟器外观时，只需把下载后的模拟器外观档桉解开成一个资料夹，再将资料夹放到「android_sdk/platforms/android- 1.5/skins」目录下。做完后在命令列中输入「android list targets」命令，即可发现 id 2 (Android 1.5) 列表中的「Skins」项目新增加了「G1」一项(可透过更改目录名称来自行命名)。

代码：

```
id: 2
    Name: Android 1.5
    Type: Platform
    API level: 3
    Skins: G1, HVGA (default), HVGA-L, HVGA-P, QVGA-L, QVGA-P
```

要建立使用新模拟器外观的模拟器，可以在之前命令后加入「--skin」选项。命令如下：

代码：

```
$ android create avd --target 2 --name devphone –skin G1
```

我们也可以偷懒不用参数的全名，将参数用简写表示。即用「-t」表示「--target」，用「-n」表示「--name」，用「-s」表示「--skin」。改输入如下：

代码：

```
$ android create avd -t 2 -n devphone –s G1
```

移除模拟器场景

要移除一个模拟器场景，直接删除在「android_sdk/platforms/android-1.5/skins」中的对应目录即可。

移除程序

我们已经顺利地启动了模拟器，那麼，该怎麼移除安装到模拟器上的程序

Android SDK 中提供一个 adb (Android Debugger) 命令行工具 (在 android-sdk/tools 中)，我们可以用裡面的 shell 工具连上模拟器来移除应用程序。在某些平台上，这些动作可能需要拥有 root 权限才能执行。

首先打开命令列，启动 adb shell

代码：

```
$ adb shell
```

接着切换到 data/app 目录中

代码：

```
$ cd data/app/
```

使用 ls 命令(等同 windows 上命令行的 dir 命令)来检视档桉列表

代码：

```
# ls
  -rw-r--r-- system    system      1325833 2007-11-11 20:59 ApiDemos.apk
```

接着使用 rm 命令来删除 ApiDemos 应用程序

代码：

```
# rm ApiDemos.apk
# ls
```

移除模拟器

我们可以使用「android list avd」命令来列出所有的模拟器

代码：

```
$ android list avd
Available Android Virtual Devices:
    Name: cupcake
    Path: /Users/mac/.android/avd/cupcake.avd
  Target: Android 1.5 (API level 3)
    Skin: HVGA
```

表示现在系统中有一个名为 cupcake 的模拟器。 我们可以使用「android delete avd --name cupcake」命令来删除名称为「cupcake」的模拟器。

代码：

```
$ android delete avd --name cupcake
AVD 'cupcake' deleted.
```

删除后再次执行「android list avd」命令，得到的结果为

代码：

```
$ android list avd
Available Android Virtual Devices:
```

表示系统中已经不存在任何模拟器，我们真的已经将模拟器删除了。

阅读参考

* 模拟器操作细节 http://developer.android.com/guide/deve ... lator.html
* 模拟器外观下载 http://www.android.encke.net/

建立一个 Android 程序

在前几章我们已经学到怎麼开启现有的专桉，也导览过了整个模拟器的设定流程。 现在我们从设计一个简单实用的身高体重指数计算(BMI)应用程序开始，学习设计一个 Android 应用程序所需的基础。

维基百科上这么介绍 BMI 身高体重指数：

身高体重指数（又称身体质量指数，英文为 Body Mass Index，简称 BMI）是一个计算值。

...当我们需要比较及分析一个人的体重对于不同高度的人所带来的健康影响时，BMI 值是一个中立而可靠的指标。

简而言之，我们要设计的程序就是允许输入身高体重，按下「计算 BMI」键后就在萤幕上显示 BMI 值，并弹出「你应该节食」、或「你应该多吃点」...等健康建议。健康建议的判断: 只要 BMI 值超过 「25」 时就算偏胖、BMI 值低于「20」 就算偏瘦。 判断写得很简单。毕竟我们要学习的关键知识，不是在于 BMI 值的算法或健康建议的内容，而是在于 Android 程序的运作方式。

参考资源 http://zh.wikipedia.org/wiki/身高体重指数

我们这就先从建立一个新的程序开始吧。

建立新工程

首先，我们照前面章节的教学，建立一个新的程序工程。并将新工程名称命名为 BMI。在「内容」栏里，我们选择「在工作区域中建立新专桉」(Create new project in workspace)。这时，如果在"选择栏"取消掉勾选「使用预设目录」(Use default location) 选项，我们就可以切换储存专桉的资料夹。大部分的时候我们并不需去改动这个选项，而是直接使用预设的资料夹。

前面章节中都是开启现有的专桉，因此那些专桉属性 (Properties) 等内容都被自动填写好了。这章中要从无到有新建一个专桉，因此我们必须自行填写专桉相关的属性。

在此对"New Android Project" 对话框中出现的这些栏位作些简单的说明：

名称 描述

Project Name 包含这个项目的资料夹的名称

Application Name 显示在应用程序上的标题

Package Name 包(Package)名称，JAVA 的习惯是用套件名称来区分不同的类别(class)。依照专桉的不同，我们会起不同的路径名称。

Create Activity 使否建立这个是项目的主要类别，勾选后可以指定这个类别的名称。这个类别是一个 Activity 类别的子类别。我们可以在「Activity」中启动程序和控制程序流程，或是根据需要控制萤幕、界面。

Build Target 选择用来编译专桉的 SDK 版本。 当选定了 Build Target 后，开发工具会在 Min SDK Version 栏位中自动填入对应的值

Min SDK Version 本应用程序所支持的最低 SDK 版本代号。

我们在栏位中分别填入以下的值：

名称 ： 值
Project Name ： BMI
Application Name ： BMI
Package Name : com.demo.android.bmi
Create Activity : Bmi
Min SDK Version : 3（自动填入）

填好值后按下「Finish」 按钮，就建立好新工程了。

注 意 Package Name 的设定，必须至少由两个部分所构成，例如:com.android。「 Activity Name」是指定用来产生预设 java 程序码的文件名称，与文件中预设 Activity 类别 (class) 的名称。依照 java 语言的命名习惯，「Activity Name」最好是用开头大写的形式。

回到 Eclipse 主画面，我们注意到在左侧 Package Explorer 视窗中已顺利新增加了一个 BMI 目录。

程序工程架构

乍看之下，Android 插件已帮我们建立了不少档案。 检视新建立的 BMI 档桉夹中的内容，我们可以将一个 Android 应用程序基本的档桉结构归纳成如下：

我们来看看 Android 应用程序的基本档桉结构，以及各自所负责的内容。
src/ 源代码(source)目录
src 目录中包含了一个 Android 应用程序中所需的各个程序码档桉。这些档桉被包在对应 package 的子目录下。(如本章的 BMI 例子中，子目录指的就是 /src/com/demo/android/bmi/)
src 目录中比较关键的程序有:
1. Bmi.java 这个档桉的档名会随着你在建立一个新的程序专桉画面中所填入「Create Activity」栏位值的不同而改变。这是新程序专桉中的主要程序区块。我们开发 Android 程序的多数时间，都是在 src 目录下和 Android 打交道。
gen/ 自动生成(Generate)目录
gen 目录中存放所有自动生成的档桉。
gen 目录中最关键的程序就是 R.java 档。
2. R.java 这个档是自动产生的。会由 ADT 插件自动根据你放入 res 目录的 XML 描述文件、图像等资源，同步更新修改 'R.java' 这个文件中。所有的 Android 程序中都会有以 R.java 为名的这个档桉，你完全不需要，也应避免手工修改 R.java 这个档桉。
R.java 中自动产生的「R」 类别就像是个字典一样，包含了使用者界面、图像、字串等各式的资源与相应的编号(id)。Android 应用程序中很多时候会需要透过 R 类别调用资源。 编译时编译器也会查看这个资源列表，没有使用到的资源就不会编译进去，为手机应用程序节省不必要没用的空间。
res/ 资源(Resource)目录
「res」 目录中存放所有程序中用到的资源档桉。"资源档桉"指的是资料档桉，或编译时会被转换成程序一部分的 XML 描述档。Android 针对放在「res」 目录下的不同子目录的资源，会有各自不同处理方式。因此我们写程序时，最好能搞清楚各目录下分别可放置的

内容。

res/ 中的程序:

3. layout/ 版面配置(layout)目录 「layout」目录包含所有使用 XML 格式的界面描述档。「layout」 中的 XML 界面描述档就像写网页时用到的 HTML 档桉一样，用来描述萤幕上的版面编排与使用的界面元件。XML 界面描述档描述的内容可以是整张萤幕，也可以只描述一部分的界面(例如描述用来产生对话框的界面元件)。

虽然你也能直接通过 Java 来建立使用者界面，不过透过 XML 描述档来建立使用者界面相对更简单，架构也更清晰，以后维护时更容易釐清问题。要使用这些界面元件，应透过「R.java」 档中自动产生的 「R」 类别来调用。

4. values/ 参数值(value)目录「values」目录包含所有使用 XML 格式的参数值描述档，可以在此添加一些额外的资源如字串(很常用)、颜色、风格等。使用时也是透过 「R」 类别来调用。

Android 功能清单

5. AndroidManifest.xml

「AndroidManifest.xml」 是 Android 程序的功能清单，应用程序在这离列出该工程所提供的功能。当应用程序开启时，会提供诸如内容提供者(ContentProvider)、处理的资料类型、 实际运行的类别、跨应用程序的资讯等等讯息。 你可以在此指定你的应用程序会使用到的服务(诸如电话功能、网路功能、GPS 功能等)。当你新增一个页面行为类别 (Activity) 时，你也需要先在此注册这个新增的 Activity 类别后，才能顺利调用。

参考资料

Android 应用程序的档桉组织方式与用途 http://developer.android.com/guide/appe ... l#filelist

初识 GUI 界面

描述使用者界面

将一份创意落实到可执行的应用程序，背后需要的是从阅读与写作程序码中累积的经验，并有坚持理念、直到完成的耐心。

表达使用者界面:

我们可以先用前几章教的方法设定并执行模拟器，看看模拟器运作后的结果：

我们看到一个文字栏位，上面有一串文字 「Hello World, Bmi!」。这就是 Android 预设程序架构的范例。

由于才刚开始实际接触到 Android 应用程序，我们先从简单的开始：这一节中，我们的目标是将 「Hello World, Bmi!」 换成别的文字。

那么，「Hello World, Bmi!」，这串字串藏在哪里呢？

先打开 「res/layout/main.xml」

代码:

```
1  <?xml version="1.0" encoding="utf-8"?>
 2  <LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
3     android:orientation="vertical"
4     android:layout_width="fill_parent"
5     android:layout_height="fill_parent"
6     >
7      <TextView
8          android:layout_width="fill_parent"
9          android:layout_height="wrap_content"
10        android:text="Hello World, Bmi"
11        />
12 </LinearLayout>
```

原来「Hello World, Bmi!」字串就藏在「res/layout/main.xml」这个档桉的第 10 行中。我们只要简单地将第 10 行修改成如下

代码:

```
android:text="Hello World, Bmi!"
```

再执行一次模拟器，就可以得到一个相似的应用程序，只是内文变成了我们刚刚修改的内容。

既然找到了「Hello World, Bmi!」字串，我们就试着将「android:text」属性值从「Hello World, Bmi!」改成「哈喽，BMI」，然后执行看看吧。

代码:

```
android:text="哈喽，BMI"
```

结果发现 Android 模拟器中文嘛也通，字型也相当漂亮。

要开始学习 Android 应用程序确实很简单吧？不过为了显示「Hello World, Bmi」，也用到了许多程序码。到底这些程序码有什么含意呢？

我们马上来学习 「main.xml」这个 XML 界面描述档的内涵吧。

Android 平台，使用者 GUI 界面都是透过 ViewGroup 或 View 类别来显示。ViewGroup 和 View 是 Android 平台上最基本的使用者界面表达单元。我们可以透过程序直接呼叫的方法，调用描绘使用者界面，将萤幕上显示的界面元素，与构成应用程序主体的程序逻辑，溷合 在一起编写。或是，也可以将界面显示与程序逻辑分离，照着 Android 提供的这个较优雅的方式，使用 XML 描述档，来描述界面元件的组织。

讲解

我们看到的「Hello World, Bmi」就包含在「main.xml」 这个档桉中。 接着，我们就直接分部份来讲解这个「main.xml」的内容：

第 1 行

代码:

```
<?xml version="1.0" encoding="utf-8"?>
```

XML (Extensible Markup Language) 是一种标记描述语言，不管是语法还是看起来的样子,都相当类似网页所使用的 HTML 标记语言。XML 被广泛地运用在 Java 程序的设定中。「main.xml」 文件裡，第一行是每个 XML 描述档固定的开头内容，用来指示这个文字档桉是以 XML 格式描述的。

第 2, 6 与 12 行

代码:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"></LinearLayout>
```

接着我们看到第一个标签，与 HTML 网页标签相当类似。

代码:

```
<LinearLayout></LinearLayout>
```

" 线性版面配置"(LinearLayout)标签，使用了两个「LinearLayout」标签，来表示一个界面元件的区块。后头的标签前加上一个「/」 符号来表示结束标签。"线性版面配置" 所指的是包含在 「LinearLayout」标签中，所有元件的配置方式，是将一个接一个元件由上而下排队排下来的意思。

代码:

```
 xmlns:android="http://schemas.android.com/apk/res/android"
```

xmlns 开头的这串叙述，是用来宣告这个 XML 描述档桉的的名称空间(NameSpace)，后面接的 URL(网址)，表示这个描述档桉会参照到 Android 名称空间提供的定义。 所有 Android 版面配置档桉的最外层标签中，都必须包含这个属性。

注意标签需要两两对称。一个标签「

引用:

```
<linearlayout>
```

」在一串叙述的前头，另一个标签「

引用:

```
</linearlayout>
```

」在叙述的末尾。 如果你修改过的标签没有闭合(忘了加 <、/、> 等符号)，Eclipse 画面上也会出现小小的警示符号来提醒你。

第 3-5 行

代码:

```
  android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
```

这些包含在「

代码:

```
<linearlayout>
```

」 标签中的叙述被称为「LinearLayout」标签的「属性」。Android 应用程序在 layout 目录中的标签，大多数的属性前都有一个「android:」前缀。同一个界面元件的属性之间，是以空白做区隔，因此事实上你也能将多个属性写在同一行。当 然，将属性排成多行更易于阅读。我们应该保持这个好习惯。

界面元件都有许多共同的属性，例如界面元件的长，宽度设定属性。Android 界面元件的宽度、长度设定属性分别叫做「android:layout_width」、「android:layout_height」。两个都设定为 「fill_parent」参数值。「fill_parent」 如其名，所表达的的意思就是"填满整个上层元件"。预设 LinearLayout 界面元件就会充满整个屏幕空间。

界面元件彼此间也会有一些不同的属性，例如「LinearLayout」(线性版面配置) 标签的「android:orientation」(版面走向) 属性。在此填入「vertical」 (垂直)属性值，表示这个界面的版面配置方式是从上而下垂直地排列其内含的界面元件。

「android.view.ViewGroup」 是各种布局配置(layout)和视图(View)元件的基础类别。常见的实现有 LinearLayout(线性版面配置)、FrameLayout(框架版面配置)、TableLayout(表格版面配置)、AbsoluteLayout(绝对位置版面配置)、RelativeLayout(相对位置版面配置)等。

虽然有这么多种版面配置方式可以选用，但大多数的应用程序并不需特地去改变预设的 LinearLayout 的配置，只要专注在其中填入需要的界面元件即可。所以从第 7 行之后的内容才是一般应用程序开发时较常修改之处。

第 7 和 11 行

代码:

```
<TextView/>
```

TextView (文字检视)是我们看到的第一个熟悉的界面元件。其作用是显示文字到屏幕上。你可能注意到这个标签结尾使用了 「/>」 符号。「/>」符号表示这个 XML 叙述中没有内文，亦即此界面元件描述中不再包含其他界面元件，也表示这个界面元件就是这个萤幕中最小的组成单元 了。

第 8-10 行

代码:

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Hello World, bmi"
```

我们来看看 TextView 界面元件中包含了哪些属性。

「android: layout_width」和「android:layout_height」我们刚刚已经学过了，分别代表宽度跟长度。「android: layout_width」 的 「fill_parent」 参数值表示宽度填满整个上层界面元件(即 LinearLayout 界面元件)。「android:layout_height」则是用上一个新的参数值「wrap_content」(包住内容)，亦即随着文字栏位行数的 不同而改变这个界面元件的高度。最后的 「android:text」 属性则是 TextView 界面元件的主要属性，亦即文字栏位中显示的文字内容。至于「@string/hello 」这段字串所代表的意义，马上会接着在后面章节说明。我们现在已知道是：只要将「android:text」属性内容替换成我们想要文字，在预览画面或在 模拟器中就会显示对应的文字。

将以上的 XML 描述综合起来，我们就可以得知「main.xml」 想表达的界面。

设计 GUI 界面

视图(View)

软件设计的过程中，常常会遇到需要频繁修改使用者界面的情境。改着改着程序设计师们就累积起了一些经验，也归纳出了许多应对之道。如着名的 MVC(Model-View-Controller)模式。Google Android 为我们考虑了界面修改问题。Android 为了单纯化界面修改方式，采用了目前比较流行的解决方桉--即将界面描述部份的程序码，抽取到程序外部的 XML 描述文件中。

我们在前面的过程种已经学到，如何在 Android 应用程序中替换 TextView 界面元件所显示的纯文字内容。那么...这个经验能不能直接用到 BMI 应用程序的设计上呢？

我们先回过头来想想， BMI 应用程序最少应该需要射门些什么元件。

为 了输入 BMI 程序所需的身高体重值，大致上我们需要两个 TextView 元件用来提示填入身高体重数字，另外也需要两个文字输入栏位用来填入身高体重数字。我们还需要一个按钮来开始计算，而计算完也需要一个 TextView 元件来显示计算结果。于是初版的 BMI 应用程序界面的样子就浮现出来了。

查阅文件

话说回来，我们从哪得知各种可用的界面元件呢？其中一个方法是查阅文件。

Android 文件网站上找到各种可用界面元件列表。

http://developer.android.com/guide/tuto ... index.html

例如我们想查看 EditText 的内容，我们可以点进 EditText 连结查看其内容。

http://developer.android.com/reference/ ... tText.html

你会看到一个详细地惊人的网页。

边举其中常用到的 EditText 界面元件为例。EditText 界面元件的作用就是提供一个文字输入栏位。EditText 继承自另一个叫 TextView 的界面元件，TextView 界面元件的作用是提供文字显示，所以 EditText 界面元件也拥有所有 TextView 界面元件的特性。 此外，文件中你也可以查找到 EditText 栏位的一些特殊属性，如「android:numeric="integer"」(仅允许输入整数数字)、「android:phoneNumber="true"」(仅允许输入电话号码)，或「android:autoLink="all"」(自动将文字转 成超连结)。 例如要限制 EditText 中只允许输入数字的话，我们可以在 XML 描述档中，透过将 EditText 的参数「android:numeric」 指定为「true」，以限制使用者只能在 EditText 文字栏位中输入数字。

离线文件

当你处于没有网路连接的情况下时，也可以找到 Android 文件参考。 在下载了 android-sdk 后，将之解压缩，你可以在「android-sdk/docs」 目录中(android_sdk/docs/reference/view-gallery.html) ，找到一份与线上文件相同的文件参考。

开始设计

我们从实例来开始，定义一个基本 BMI 程序所需的身高(Height)输入栏位，就会用到 EditText，与 TextView 界面元件，其描述如下：

代码：

```
1  <TextView
2     android:layout_width="fill_parent"
3     android:layout_height="wrap_content"
4     android:text="身高 (cm)"
5     />
6  <EditText android:id="@+id/height"
7     android:layout_width="fill_parent"
8     android:layout_height="wrap_content"
9     android:numeric="integer"
10    android:text=""
11    />
```

可以看到 EditText 界面元件描述的基本的组成与 TextView 界面元件相似，都用到了「android:layout_width」与「android:layout_height」属性。另外，指定的另外两个属性 「android:numeric」、「android:text」则是 EditText 界面元件的特别属性。「android:text」属性是继承自 TextView 界面元件的属性。

代码：

```
android:numeric="integer"
android:text=""
```

将「android:numeric」 指定为 「integer」，可以限制使用者只能在 EditText 文字栏位中输入整数数字。「android:text」属性则是指定 EditText 界面元件预设显示的文字(数字)。

我们再来看看 Button (按钮)界面元件

代码：

```
<Button android:id="@+id/submit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="计算 BMI 值"
        />
```

Button 界面元件同样有 「android:layout_width」与「android:layout_height」属性，另外一个「android:text」属性则用来显示按钮上的文字。

整合

我们这就从文件中挑出我们需要的 TextView(文字检视)、EditText(编辑文字)、Button(按钮) 三种界面元件，照前面的设计摆进 LinearLayout (线性版面配置)元件中。完整的「main.xml」界面描述档如下：

代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
```

```
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="身高 (cm)"
        />
    <EditText android:id="@+id/height"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:numeric="integer"
        android:text=""
        />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="体重 (kg)"
        />
     <EditText android:id="@+id/weight"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:numeric="integer"
        android:text=""
        />
     <Button android:id="@+id/submit"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="计算 BMI 值"
        />
     <TextView android:id="@+id/result"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text=""
        />
         <TextView android:id="@+id/suggest"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text=""
           />
</LinearLayout>
```

我们可以启动模拟器检视执行结果。或是在页面标签下选择「Layout」标签，来预览页面配置。

启动模拟器之后，模拟器画面上出现了两个输入栏位。栏位上方分别标示着「身高(cm)」、「体重 (kg)」。在两个输入栏位下方，是一个标示着「计算 BMI 值」的按钮。 当你在栏位中试着输入文字或数字(你可以直接用电脑键盘输入，或按着模拟器上的虚拟键盘输入)时，你也会发现，XML 描述档的描述中对两个 EditText 栏位所规定的，栏位中只能输入数字。

我们在上面 XML 描述档中定义的最后两个 TextView 界面元件，由于并未替这两个界面元件指定「android:text」属性，所以在萤幕上并未显示。这两个界面元件在后面章节中会派上用场。

革命的路还长

高兴了没多久，你发现按下"计算 BMI 值" 按钮后，应用程序完全没反应。这是正常的，因为我们还没处理从界面输入取得身高体重、将数值导入 BMI 计算方式、将结果输出到萤幕上...等等 BMI 应用程序的关键内容。不过在进入了解程序流程之前，我们还有一个「android:id」属性尚未解释。 接着我们将透过讲解「android:id」属性，来进一步了解 Android UI。

视觉化的界面开发工具

目前的 ADT 版本提供了预览界面的功能，但尚未提供方便地视觉化拖拉界面元件的开发工具。以后也许 ADT 会加入完整的 GUI 拖拉设计工具。

但在 ADT 加入完整的 GUI 拖拉设计工具之前，已经有人写出来了对应 Android 的 GUI 拖拉设计工具，可供使用。

DroidDraw - Android GUI 拖拉设计工具

http://code.google.com/p/droiddraw/

来源:http://code.google.com/p/androidbmi/wiki/BmiUI

获取标识 ID

存取识别符号

在上一章谈了 XML 描述档中界面元件的各种「android:」开头的属性。要使用一个界面元件，第一件事就是定义出界面描述档。大部分的界面元件(如 LinearLayout、TextView)不需要在程序中作后续处理，因此可以直接描述。不过对于那些将在程序中被参考(reference)到的界面元件（如按钮 Button、文字输入栏位 EditText），我们需要透过在 XML 描述档中，定义该界面元件的「android:id」识别符号属性。之后在程序中所有与这个界面元件有关的操作，都能根据「android:id」 识别符号来在调用这个界面元件。

代码：

```
<EditText android:id="@+id/height"     />
```

前面章节提过，写作时最好将 XML 描述档属性分行列出，以易于阅读(增加可读性)。而我们的范例却将 android:id 属性直接摆在 EditText 标签后。其实这么做同样是基于易于阅读的考量。当然你也可以将「android:id」属性分行列出，或是将「android:id」属性放在属性列表的 中间或最后头，这些作法都是允许的，本书中一律使用将 android:id 属性直接摆在界面元件标签后的写法。

android:id 属性的内容长得比较特别：

代码：

```
@+id/height
```

「height」是这个界面元件的 android:id。以后的程序中会使用「R.id.height」来取得这个界面元件。「@+id」的意思是我们可以通过这个识别符号来控制所对应的 界面元件，「R」类别会自动配置一个位址给这个界面元件。 「R」类别的内容则可以透过查看 R.java 得知。

XML 描述档与 R.java 档

在 Android 系统中，我们使用 XML 来定义 UI。但是有些稍微有经验的开发者可能会有疑问：

「用 XML 来描述界面固然方便，但是对于手机程序来说，直接用 XML 档桉是不是太占空间了？」。

没错，如果 Android 是直接使用 XML 来储存界面描述到手机上的话，一定会使用比起现在大的多的档桉空间。解决的方法是 Android 并不直接使用 XML 档桉，而是透过 Android 开发工具，自动将 XML 描述档转换成资源档桉。一旦应用程序要使用某个界面或是任何种类的资源(字串、图片、图示、音效...)，都使用索引来查询。

当你建立一个 BMI 新专桉，打开位于 「src/com/demo/android/bmi」 目录下的「R.java」档，你可以看到如下的程序码：

代码：

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */
```

```
package com.demo.android.bmi;

public final class R {
        public static final class attr {
        }
        public static final class drawable {
            public static final int icon=0x7f020000;
        }
        public static final class layout {
            public static final int main=0x7f030000;
        }
        public static final class string {
            public static final int app_name=0x7f040000;
        }
    }
```

在照着前一章新增了 XML 描述后，再次打开打开 「src/com/demo/android/bmi」 目录下的 「R.java」 档 ，你可以看到如下的程序码：

代码：

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */

package com.demo.android.bmi;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int height=0x7f050000;
        public static final int result=0x7f050003;
        public static final int submit=0x7f050002;
        public static final int suggest=0x7f050004;
        public static final int weight=0x7f050001;
    }
    public static final class layout {
```

```
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
    }
}
```

我们看到在 R.java 档桉中，分别有 attr（属性）、drawable（图片、图示）、id（识别符号）、layout（界面描述）、string（文字）这几种资源型态，就 XML 描述档中的 id 来说，开发工具会根据 XML 描述档中指定的 id，生成对应的资源，并自动指定一个位址。

Google 官方文件是这么解释「R.java」档案的作用的：

A project's R.java file is an index into all the resources defined in the file. You use this class in your source code as a sort of short-hand way to refer to resources you've included in your project. This is particularly powerful with the code-completion features of IDEs like Eclipse because it lets you quickly and interactively locate the specific reference you're looking for.

The important thing to notice for now is the inner class named "layout", and its member field "main". The Eclipse plugin noticed that you added a new XML layout file and then regenerated this R.java file. As you add other resources to your projects you'll see R.java change to keep up.

有了「R.java」做中介，在 XML 描述档中，我们可以透过

@[类型]/[识别符号]

这样的语法来为某个界面元件提供识别符号，以供程序控制。

例如，我们可以用 「@+id/height」来为对应供输入身高数字的 EditText 元件提供识别符号。

将字串抽离 XML

当我们在 res 资料夹中新增各种一个 XML 档桉，或是一张图片时，开发工具会从 res 资料夹中手机集，并将各种资源整成一个索引，自动产生出 R.java 档。透过这个特性，我们可以进一步加工我们的 XML 描述档，让界面更易于维护。开启 res/values/strings.xml，原始的内容为

代码：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">BMI</string>
    </resources>
```

上面只定义了一个字串「app_name」，用来表示应用程序名称（在之后讲解 AndroidManifest.xml 档桉时将会用到）。 我们看到表示字串的格式为

代码：

```
<string name="识别代号">文字叙述</string>
```

我们将上一章中的叙述抽取出来，整理进 strings.xml 档桉。

完整的 strings.xml 档桉如下：

代码：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">BMI</string>
        <string name="height">身高 (cm)</string>
        <string name="weight">体重 (kg)</string>
        <string name="bmi_btn">计算 BMI 值</string>
        <string name="bmi_result">你的 BMI 值是 </string>
    </resources>
```

在 strings.xml 档桉中，我们在原本的 app_name 字串外，自行定义了另外几个字串。如果我们再次开启 「R.java」档，我们会发现档桉中的 string 类别中也自动索引了上面定义好的字串：

代码：

```
    public static final class string {
        public static final int app_name=0x7f040000;
            public static final int bmi_btn=0x7f040003;
            public static final int bmi_result=0x7f040004;
            public static final int height=0x7f040001;
            public static final int weight=0x7f040002;
        }
```

接着，我们把这些字串应用到之前定义好的 XML 描述档中。透过使用@string/[识别符号]这样存取 string 类型的格式，来取代 main.xml 档桉中原本写死的文字叙述。

完整的程序码如下：

代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
    android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        >
        <TextView android:layout_width="fill_parent"
                  android:layout_height="wrap_content"
                  android:text="@string/height"
                  />
        <EditText android:id="@+id/height"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:numeric="integer"
            android:text=""
            />
        <TextView
            android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:text="@string/weight"
        />
    <EditText android:id="@+id/weight"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:numeric="integer"
        android:text=""
        />
    <Button android:id="@+id/submit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/bmi_btn"
        />
    <TextView android:id="@+id/result"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text=""
        />
    <TextView android:id="@+id/suggest"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text=""
        />
</LinearLayout>
```

再次运行 Android 模拟器，我们看到与前一节完全相同的界面。但就界面而言，透过将描述字串统一集中在 「string.xml」 中，我们以后要修改界面时更有弹性了。

至此我们已经完成了 BMI 应用程序负责「显示 (View)」的部份。

新增 XML 文件

我们在前面都只修改到开发工具帮我们产生的档桉，而事实上，我们所有在「res」目录中所做的修改，开发工具都会自动搜寻，将之整理到「R.java」中。因此我们也可以在「src/values」中建立自己的文字描述档桉。

我们这就在「res/values」目录中建立一个 「advice.xml」档，上面将包含 BMI 程序算出 BMI 值后将给予的建议文字，完整的档桉如下：

代码:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="advice_light">你该多吃点</string>
        <string name="advice_average">体型很棒喔</string>
        <string name="advice_heavy">你该节食了</string>
    </resources>
```

打开「R.java」档，我们发现「advice_light」、「advice_average」、「advice_heavy」也已整理进「R.java」档的索引中，以供程序调用。

那么接下来，我们就开始进入到了解 Android 程序流程的部分吧。

解读程序流程

解读程序流程

接着要观察主要程序逻辑的内容。打开 "src/com/demo/android/bmi" 目录下的"Bmi.java"档桉，Eclipse+Android 开发工具已经帮我们预先建立好了基本的程序逻辑。其预设的内容如下：

代码：

```
1  package com.demo.android.bmi;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class Bmi extends Activity {
7      /** Called when the activity is first created. */
8      @Override
9      public void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.main);
12     }
13 }
```

讲解

比起什么标签都对称的 XML 界面描述档来说，这个以 Java 程序语言写成的档桉虽然篇幅短，但反而要难读得多。

我们将程序直接拆开，分成几个部份来讲解这个"Bmi.java" 档桉的内容：

第 1 行:

代码：

```
package com.demo.android.bmi;
```

这一行的作用是指出这个档桉所在的名称空间。"package"(包)是其关键字。使用名称空间的原因是程序一旦扩展到扩展到某个大小，程序中的变数名称、 方法名称、类别名称难免重複，这时就可以将定义的名称区隔管理在 package 下，以避免相互冲突的情形发生。Java 的 package 设计成与档桉系统结构相对应，如我们的 package 设定是 "package com.demo.android.bmi"，则这个类别就该在指定目录的"com/demo/android/bmi"路径下可以找到。

同时也别忘了 Java 程序语言每段叙述语句的结尾处，与大部分的程序语言一样需加上一个分号";"，以表示一行程序叙述的结束。

第 3,4 行:

代码：

```
import android.app.Activity;
import android.os.Bundle;
```

程序中预设导入了 "android.app.Activity"跟"android.os.Bundle"两个 Package，在所有的 Android 应用程序中都会用到这两个 Package。"import"(导入)是用作导入 Package 的关键字。在 Java 语言中，使用到任何 API 前都要事先导入相对应的 Package。我们马上将学到这两个 Package 的用途。

Android 支援的 Package 与标准的 Java(j2se) 不尽相同。在写 Android 应用程序时，你偶而可能需要参考可用的 API 列表，以确认使用到的 Package 是否有内建支援。后续章节中也将讲解如何透过新增"jar"档来呼叫额外的 Package。

完整的 API 可查阅官方的 package 列表：

http://code.google.com/android/reference/packages.html

第 6,13 行:

代码:

```
public class Bmi extends Activity {
}
```

第 6 行开始了程序的主体。其组成是这样的：

代码:

```
public class Bmi
```

"Bmi"是这个类别的名称。"class"则是用作宣告类别关键字。"public"关键字是用来修饰"Bmi"这个类别。表示"Bmi"是个"公开"的类别，可以从 package 外部取用。

"public class Bmi"后面再加上"extends Activity"叙述，则表示 "Bmi"这个类别的功能、型别等全继承自"Activity"类别。"extends"是继承(Inherit)类别的关键字。"Activity"是 来自于我们在第 3 行刚导入的 Package。

因此整句话的含意即："宣告一个公开的 Bmi 类别。这个 Bmi 类别继承了程序开头导入的 Activity 类别"。

"{}"大括号规范了一个程序区块。大括号中的程序表达的这个程序区块的主要内容。

第 7 行:

代码:

```
/** Called when the activity is first created. */
```

第 7 行提供了位于其下的函式的注释。"/* */" 是 Java 语言的多行注解符号，位于其中的文字内容不会被编译。"/*"叙述后多出来的一个"*"号被视为内文。顺便提醒一下，Java 程序语言中两个斜线"//"表示的是单行注解符号。单行注解符号"//"与多行注解符号"/* */"不同的地方是，只有与"//"符号同行的文字才不会被编译。

第 8-9, 12 行:

代码:

```
@Override
```

```
public void onCreate(Bundle savedInstanceState) {
}
```

第 9 行开始了这个方法(Method)的主体。其组成是这样的：

代码：

```
public void onCreate(Bundle savedInstanceState) {
}
```

"onCreate"是这个方法的名称。"void"则是宣告了这个方法的回传值的型别(type)。"public"关键字是用来修饰"onCreate"这个方法。表示"onCreate"是个"公开"的方法，可以由 bmi 类别外部取用。

方法的回传值的型别，即是这个方法的型别。"onCreate"这个方法使用"void"型别，表示"onCreate"这个方法不需回传值。

同时，这个方法传入了一个名为"savedInstanceState"的"Bundle"型别参数，"Bundle"型别正是来自我们前面所导入的 Package 之一。我们并不需要知道太多"Bundle"型别或"savedInstanceState"实体的细节，只要知道"Bundle"的内容与手机平台的记 忆体管理有关。

当 Android 应用程序启动、换到背景等待、关闭时，都会用到 "savedInstanceState"这个实体来处理记忆体相关的事宜。当然，你也可以用其他名称来代替它。还好"onCreate"这个方法永远都 是传入"Bundle savedInstanceState"这个参数，写应用程序时只要正确照规定传入即可，你可以不用太去在意它。

给对 Bundle 是什么有兴趣的读者：

"Bundle"可以保存程序上一次关闭(冻结)时的状态。你可以透过覆写 onFreeze 方法(与 onCreate 方法的作用类似) 来保存冻结前的状态。 当程序启动（Activity 重新初始化）时，会再次呼叫 onCreate 方法，你就能从 savedInstanceState 中得到前一次冻结的状态。我们也可以透过"Bundle"来将这个 Activity 的内容传到下一个 Activity 中。 之后讲 Activity 时，也会讲解 onCreate/onFreeze 等方法的关系。

"{}"大括号规范了一个程序区块。大括号中的程序表达 onCreate 这个程序区块的主要内容。

代码：

```
@Override
public void onCreate(Bundle savedInstanceState)
```

从前面的讲解中，我们学到了在任何一个 Android 专桉目录里，只要打开"Referenced Libraries"目录的"android.app" 分类，都可以找到"Activity.class"这个类别。现在我们再深入一些查看"Activity.class" 类别。你要做的，只是依照图示，找到 Android 工具中的 "Referenced Libraries" 目录，从"android.app"分类里找到"Activity.class"类别，并按下 "Activity.class"类别左侧的三角形图示，如 此即可展开这个类别的属性/方法列表。

我们在 Activity 类别的属性/方法列表中，发现了现在正要讲解的 onCreate 方法 (Method)。

因为"bmi" 类别继承自 Activity 类别，所以预设"bmi" 类别中其实已经有"onCreate"方法了。

事实上，"onCreate" 方法正是每个 Activity 类别初始化时都会去呼叫的方法。"@"开头的语句表示装饰子(decorator)语句，"@Override"语句的作用是告诉程序我们要覆写这

个"onCreate"方法。当我们打开程序时，程序不再使用从"bmi" 类别中继承来的"onCreate"方法，而是使用我们在程序中自订的行为。

代码:

```
@Override
public void onCreate(Bundle savedInstanceState) {
}
```

我们讲解了整段程序，其含意是"覆写 bmi 类别中公开的 onCreate 方法。这个"onCreate"方法无回传值型别，而且这个方法传入了一个名为 "savedInstanceState" 的 Bundle 型别参数。现在来看看"onCreate"方法中包含的程序内容。

第 10, 11 行:

代码:

```
super.onCreate(savedInstanceState);
```

"super"是关键字。代表着这个 "Bmi"类别的上层类别(Activity)。
"super.onCreate(savedInstanceState);"的意思就是："执行 Activity 类别中 onCreate 方法的内容"。这麼做的目的是什么呢？

Google Android 将其应用程序的界面称为视图(View)，而负责控制各种动作行为的程序主体(Controller)，则称为活动(Activity)。因此一个 Android 应用程序，必定会对应到一个以上的 Activity。 "onCreate" 方法则是每个 Activity 类别初始化时都会去呼叫的方法。我们想做的事，是保持原本"onCreate" 方法预设的动作，然后在其中加入我们想要的内容。

而 Android 产生的程序预设却覆载(@Override)了"Bmi" 类别的"onCreate" 方法。原本继承自"Activity"类别的"onCreate"方法，其原本内容都被覆载掉了。因此想将原本的"onCreate"方法内容保留，并在 其中加入我们的内容的话，就要使用"super"语句。当程序运行到我们覆写的"onCreate"方法时，透 过"super.onCreate(savedInstanceState);"语句，会先将原本"Activity"类别中的"onCreate"方法 执行一次，然后再执行我们覆写的"onCreate"方法里面其他的程序内容。

我们要执行原本的"onCreate"方法时，仍然需要提 供原本"onCreate"方法所需的传入参数。因此"super.onCreate(savedInstanceState);"语句中，我们 将"savedInstanceState"这个参数传入原本的"onCreate"函式中。"savedInstanceState"是我们 在"public void onCreate(Bundle savedInstanceState)"语句中所宣告的传入参数。

代码:

```
setContentView(R.layout.main);
```

透过萤幕显示的各种元素是按照界面层次结构来描述的。要将一个显示元素的层次结构转换显示到一个萤幕上，Activity 会呼叫它用来设定 View 的 "setContentView" 方法，并传入想引用的 XML 描述文件。当 Activity 被启动并需要显示到萤幕上时，系统会通知 Activity，并根据引用的 XML 文件叙述来描绘出使用者界面。上一章中我们定义好的 res/layout/main.xml 描述档，就是透过这个机制绘出到萤幕上。

setContentView 方法也可以在 Activity 类别中找到。

你可能也注意到 "setContentView" 方法确实是透过 "R.layout.main"来引用 XML 文件描述档的资源，而不是直接透过 res 目录来引用。

完成 BMI 程序

完成 BMI 程序

至此，我们已经完成了 bmi 程序的界面设计，并且理解了了新建立的程序。剩下我们要做的，只剩下为 BMI 程序加上程序逻辑。

很幸运的是，BMI 程序中用到的并不是什么神秘的演算法，你甚至可以透过搜寻引擎找到中文的范例。

完整的程序如下：

代码：

```
1  package com.demo.android.bmi;
2
3  import java.text.DecimalFormat;
4
5  import android.app.Activity;
6  import android.os.Bundle;
7  import android.view.View;
8  import android.view.View.OnClickListener;
9  import android.widget.Button;
10 import android.widget.EditText;
11 import android.widget.TextView;
12
13 public class Bmi extends Activity {
14     /** Called when the activity is first created. */
15     @Override
16     public void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.main);
19
20         //Listen for button clicks
21         Button button = (Button)findViewById(R.id.submit);
22         button.setOnClickListener(calcBMI);
23     }
24
25     private OnClickListener calcBMI = new OnClickListener()
26     {
27         public void onClick(View v)
28         {
29             DecimalFormat nf = new DecimalFormat("0.00");
30             EditText fieldheight = (EditText)findViewById(R.id.height);
31             EditText fieldweight = (EditText)findViewById(R.id.weight);
32             double height = Double.parseDouble(fieldheight.getText().toString())/100;
33             double weight = Double.parseDouble(fieldweight.getText().toString());
34             double BMI = weight / (height * height);
```

```
35
36              TextView result = (TextView)findViewById(R.id.result);
37              result.setText("Your BMI is "+nf.format(BMI));
38
39              //Give health advice
40              TextView fieldsuggest = (TextView)findViewById(R.id.suggest);
41              if(BMI>25){
42                  fieldsuggest.setText(R.string.advice_heavy);
43              }else if(BMI<20){
44                  fieldsuggest.setText(R.string.advice_light);
45              }else{
46                  fieldsuggest.setText(R.string.advice_average);
47              }
48          }
49      };
50 }
```

我们会学到：导入其他用到的模组、如何取得界面元件、如何对按钮设定动作。

从上面的完整程序中，我们看到上面介绍到的程序主体都还在，不过也增加了一些内容。这些内容即我们的主要程序逻辑。

讲解

//Listen for button clicks

两个反斜线是 java 语言所支援的另一种注解方式。

按钮

代码：

```
Button button = (Button)findViewById(R.id.submit);
```

宣告一个 button 实体，透过 findViewById 方法，从资源档中取得对应的界面元件(按钮)。这边取出的是"R.id.submit"按钮元件。

"R.id.submit"对应到 XML 描述档的资源是

代码：

```
<Button id="@+id/submit"/>
```

为了确保宣告的型别跟 XML 描述档中描述的界面元件型别一致，好使程序运作正常，我们在 "findViewById" 方法前加上 "(Button)"修饰，强制将取得的资源型别设成 "button" 型别。

代码：

```
button.setOnClickListener(new OnClickListener(){});
```

这句包含了" button.setOnClickListener"与其中的 "OnClickListener"两个类别。"setOnClickListener"是 button （按钮）实体的方法。

编辑栏位

代码：

```
EditText fieldheight = (EditText)findViewById(R.id.height);
```

EditText fieldweight = (EditText)findViewById(R.id.weight);

与上面 button 的宣告类似，只是改成宣告 EditText 实体，透过 findViewById 方法，从资源档中取得对应的文字栏位元件。这边取出的是"R.id.height"和"R.id.weight"文字栏位元件。

运算

代码:

```
double height = Double.parseDouble(fieldheight.getText().toString())/100;
double weight = Double.parseDouble(fieldweight.getText().toString());
double BMI = weight / (height * height);
```

BMI 值的算法是"体重除以身高的平方"。 用计算式来表示，就是

体重(weight) / 身高(height)*身高(height)

这麼看来，上面的程序码就很清晰了。我们先从身高栏位(fieldheight)、体重栏位(fieldweight)中取得使用者输入的身高体重数字，再 定义一个双倍精度幅点数(double) 型态的变数 BMI 来储存运算的结果，因此，BMI 变数中储存了运算出来的实际 BMI 值。

显示结果

我们把 BMI 值运算出来了，接着要将结果显示回屏幕上。

代码:

```
TextView result = (TextView)findViewById(R.id.result);
```

为了将结果显示到屏幕上，在之前 XML 定义档中我们已经预留了一个名为"result"的 TextView 栏位。在程序码中，我们再宣告一个 TextView 实体，透过 findViewById 方法，从资源档中取得对应的界面元件(文字显示)。这边取出的就是"R.id.result"界面元件。

代码:

```
DecimalFormat nf = new DecimalFormat("0.00");
result.setText("Your BMI is "+nf.format(BMI));
```

透过 java 内建的 DecimalFormat 函式，我们可以将运算结果，以适当的格式显示。透过"setText"方法，我们可以将指定的字串，显示在屏幕上文字类型的界面元件中。

显示建议

"显示建议"的方式与"显示结果"完全相同，只有多加了几个"if"判断式：当 BMI 值大于 25 显示过重，小于 20 显示过轻。程序码留给读者自习。

代码:

```
//Give health advice
TextView fieldsuggest = (TextView)findViewById(R.id.suggest);
if(BMI>25){
    fieldsuggest.setText(R.string.advice_heavy);}
    else if(BMI<20){
        fieldsuggest.setText(R.string.advice_light);
    }else{
        fieldsuggest.setText(R.string.advice_average);
    }
```

完整的 BMI 程序动作流程，就是使用者在身高体重栏位中输入好身高体重后，按下"计算 BMI 值"按钮，程序根据识别符号，从对应的身高体重栏位读取输入的值，并做计算，最后将计算的结果与建议显示到屏幕上。

初级

重构程序

伟大的创意少之又少，多数时候只是一些小改进。小的改进也是好的。

什么是重构

可以运作的程序跟可以维护的程序之间，还有一道难以言说的鸿沟。

一个程序设计之初，是用来解决特定问题。就像在前面章节的学习中，我们也已经写好了一个可以运作的 BMI 程序。但是对程序设计来说，当我们写越多程序，我们会希望可以从这些程序之中，找到一个更广泛适用的法则，让每个程序都清晰易读，从而变得更好修改与维护。

让程序清晰易读有什么好处呢？当一段程序被写出来，之后我们所要做的事，就是修改它与维护它。一旦程序越长越复杂，溷乱到无法维护的境界时，就只好砍掉重 练。 所以若我们能透过某些方式，例如重新组织或部分改写程序码，好让程序容易维护，那麼我们就可以为自己省下许多时间，以从容迎接新的挑战。

我们回过头来看看前面所写的 Android 程序。Android 平台的开发者已经先依照 MVC 模式，为我们将显示界面所用的 XML 描述档、显示资源所用的 XML 描述档从程序码中区隔开来。将与程序流程无关的部份分开来组织，让程序流程更清楚，相对易于维护。

而在主要程序码(Bmi.java)方面，虽然程序码量很少，还算好读，但整体上并不那麼令人满意。例如，假使我们要在这段程序码中再多加上按键、适用于多种萤幕显示模式、或是再加入选单等等内容，很快地程序码就开始变得复杂，变得不容易阅读，也开始越来越不容易维护。

因此，在继续新的主题之前，我们先来重构这个 BMI 应用程序。在重构的过程中，也许我们能学到的东西，比学任何新主题还重要呢。

MVC

我们打算重构 BMI 程序的部份 java 程序码。既然我们已经照着 Android 平台的作法，套用 MVC 模式在我们的程序组织上，那麼，我们不妨也试着套用同样的 MVC 模式在 Bmi.java 程序码上。

如何套用 MVC 模式到 Bmi.java 程序码上呢？

原来的程序片段是这样的

代码:

```
1  @Override
2  public void onCreate(Bundle icicle) {
3      super.onCreate(icicle);
4      setContentView(R.layout.main);
5
6      //Listen for button clicks
7      Button button = (Button) findViewById(R.id.submit);
8      button.setOnClickListener(calcBMI);
9  }
```

上面的程序片段中，包含了所有 Android 程序共用的标准内容， 整个程序的大致架构在前面章节中已经讲解过，现在我们从中取出我们感兴趣的部分来讨论：

代码：

```
Button button = (Button) findViewById(R.id.submit);
button.setOnClickListener(calcBMI);
```

在第 7 行我们看到一段程序码来宣告按钮物件，与针对该按钮物件作动作的程序码。button.setOnClickListener 程序码的意义是指定一个函式，来负责处理"按下"这个"按钮"后的动作。

我们可以想像，在同一个画面中，多加入一些按钮与栏位后，"onCreate" 这段程序将变得臃肿，我们来试着先对此稍作修改：

首先，我们可以套用 MVC 模式，将宣告界面元件(按钮、数字栏位)、指定负责函式等动作抽取出来，将 onCreate 函式改写如下

代码：

```
@Override
public void onCreate(Bundle icicle) {
     super.onCreate(icicle);
     setContentView(R.layout.main);
     findViews();
     setListensers();
     }
```

接着我们将宣告界面元件的部份写成一个独立的"findViews"函式：

```
     private Button calcbutton;
     private EditText fieldheight;
     private EditText fieldweight;

     private void findViews()
     {
          calcbutton = (Button) findViewById(R.id.submit);
          fieldheight = (EditText) findViewById(R.id.height);
          fieldweight = (EditText) findViewById(R.id.weight);
          }
```

顺便将原本很没个性的按钮识别参数"button"改名成"calcbutton"，以后在程序中一看到"calcbutton"，就知道是一个按下后将开始处理计算工作的按钮。

同样地，我们也将指定特定动作(按按钮)的负责函式独立出来：

代码：

```
//Listen for button clicks
private void setListensers() {
     calcbutton.setOnClickListener(calcBMI);
     }
```

如此一来，我们就将程序逻辑与界面元件的宣告分离开来，达成我们重构的目的。

完整程序如下：

代码：

```
package com.demo.android.bmi;
import java.text.DecimalFormat;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class Bmi extends Activity {
    /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle icicle) {
            super.onCreate(icicle);
            setContentView(R.layout.main);
            findViews();
            setListensers();
        }

    private Button button_calc;
    private EditText field_height;
    private EditText field_weight;
    private TextView view_result;
    private TextView view_suggest;

    private void findViews()
    {
        button_calc = (Button) findViewById(R.id.submit);
        field_height = (EditText) findViewById(R.id.height);
        field_weight = (EditText) findViewById(R.id.weight);
        view_result = (TextView) findViewById(R.id.result);
        view_suggest = (TextView) findViewById(R.id.suggest);
    }
    //Listen for button clicks
        private void setListensers() {
        button_calc.setOnClickListener(calcBMI);
    }
    private Button.OnClickListener calcBMI = new Button.OnClickListener()
    {
        public void onClick(View v)
        {
            DecimalFormat nf = new DecimalFormat("0.0");
            double height =
Double.parseDouble(field_height.getText().toString())/100;
```

```
            double weight =
Double.parseDouble(field_weight.getText().toString());
            double BMI = weight / (height * height);

            //Present result
            view_result.setText(getText(R.string.bmi_result) + nf.format(BMI));

            //Give health advice
            if(BMI>25){
                view_suggest.setText(R.string.advice_heavy);
            }else if(BMI<20){
                view_suggest.setText(R.string.advice_light);
            }else{
                view_suggest.setText(R.string.advice_average);
            }
        }
      };
    }
```

同样是"calcBMI" 函式，在完整程序中，改将"calcBMI" 函式从原本的"OnClickListener"宣告成 "Button.OnClickListener"。这个改变有什么差别呢？

阅读原本的程序码，在汇入(import)的部分可以看到，"OnClickListener"是来自于"android.view.View.OnClickListener"函式：

代码：

```
import android.view.View.OnClickListener;
```

改成"Button.OnClickListener"后，"Button.OnClickListener"就变成来自 于"android.widget.Button"中的"OnClickListener"函式，在查阅程序时，整个"Button" 与"OnClickListener"之间的关係变得更清晰。

另外，我们偷偷将"OnClickListener"中其他会存取到的界面元件识别参数，也补进findViews 宣告中：

代码：

```
private void findViews()
{
    button_calc = (Button) findViewById(R.id.submit);
        field_height = (EditText) findViewById(R.id.height);
        field_weight = (EditText) findViewById(R.id.weight);
        view_result = (TextView) findViewById(R.id.result);
        view_suggest = (TextView) findViewById(R.id.suggest);
}
```

同时，我们也把识别参数的命名方法做了统一：按钮的识别参数前加上 "button_"前缀，可输入栏位的识别参数前加上"field_"前缀，用作显示的识别参数前则加上"view_"前缀。将变数名称的命名方法统一有 什么好处呢？好处在于以后不管是在命名新变数，或是阅读程序码时，都能以更快速度命名或理解变数的意义，让程序变得更好读。

我们也把原本在程序中直接写进的字串

代码：

```
TextView result = (TextView) findViewById(R.id.result);
result.setText("Your BMI is "+nf.format(BMI));
```

改写成

代码：

```
//Present result
view_result.setText(getText(R.string.bmi_result) + nf.format(BMI));
```

并将"TextView view_result"宣告改放到 findViews 中一次处理好。

现在，整个程序流程是不是清爽了许多呢？

加入对话框

我们的程序主功能已经完成了，现在我们要试着让它看起来更像一个完整的应用程序。

接下来的几章，我们要为"BMI"应用程序加上一个选单。选单里面有一个"关于.."选项。按下"关于..."选项后，会弹出一个对话框，里面会显示"BMI"程序的相关讯息。

本章中将先学习如何处理对话框。

在本章中，我们要产生一个应用程序中常见的"关于"页面。 应用程序的"关于"页面中，通常要包含版本讯息、作者、联络方式、首页等资讯。

我们的"关于"页面将以弹出对话框的方式表现。所需要做的，是撰写负责处理对话框的"openOptionsDialog"函式，并将之附加在原本应用程序 中"calcBMI"这个按钮元件的"OnClickListener"方法上。当我们按下"计算 BMI 值"按钮时，即弹出对话框。

等我们学会了对话框的写法，在接下来学习 Android 选单的章节中，我们会改将对话框函式放入选单中。

对话框中所能显示的内容千变万化。对 Android 来说，对话框也是一种显示内容(View)。与一般全页面显示的不同之处，在于对话框会重叠显示到原本的呼叫页面上，而且在对话框的主要显示内容下方，可 能还会再附加上几个按钮，用以回到原页面，或是用来执行其他的动作。

要在 Android 程序中呼叫一个对话框，有二个主要步骤：

定义呼叫点 # 实作对话框

定义呼叫点

修改"Bmi.java"

代码：

Bmi.java

```
1  private OnClickListener calcBMI = new OnClickListener()
2  {
3      public void onClick(View v)
4      {
.          ....
6           }else{
7               view_suggest.setText(R.string.advice_average);
8           }
9           openOptionsDialog();
```

我们在"calcBMI"函式的尾端加入一行"openOptionsDialog();"，用以在每次计算完 BMI 值并显示建议后，顺便呼叫"关于"对话框。

实作对话框

紧接着"calcBMI"这个"OnClickListener"函式之后，我们实际开始撰写对话框函式。

代码：

```
private void openOptionsDialog() {
    new AlertDialog.Builder(Bmi.this)
        .setTitle("关于 Android BMI")
        .setMessage("Android BMI Calc")
        .show();
```

我们来分析这个对话框程序。

首先，显示一个最基本的对话框所需的程序码如下。

代码：

```
new AlertDialog.Builder(Bmi.this).show()
```

我们建立了一个 AlertDialog 对话框类别实体，AlertDialog 呼叫 Builder 方法来预备对应的界面元件。最后使用 show() 方法来将对话框显示在屏幕上。

透过

代码：

```
.setTitle("关于 Android BMI")
```

我们设定了对话框的标题。

透过

代码：

```
.setMessage("Android BMI Calc")
```

我们设定了对话框的主要内容。

重构

我们把其中用到的字串抽取出来，整理到"res/values/strings.xml"中。

res/values/strings.xml

代码：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    ....
    <string name="about_title">关于 Android BMI</string>
    <string name="about_msg">Android BMI Calc\n
                作者 gasolin\n\n
                gasolin+android [at] gmail.com</string>
    ....
    </resources>
```

于是 openOptionsDialog 函式变成这样：

代码：

```
private void openOptionsDialog() {
    new AlertDialog.Builder(Bmi.this)
            .setTitle(R.string.about_title)
            .setMessage(R.string.about_msg)
            .show();
```

打开模拟器，在按下按钮后，我们看到计算出 BMI 值的同时，屏幕上也弹出了一个有标题的对话框。

加入按钮

目前的对话框中，并没有提供离开对话框的方法。所以我们得按下 "Undo"按钮来离开对话框，有点不便，所以我们来为这个对话框加入一个"确认"按钮。

代码：

```
.setPositiveButton(″确认″,
    new DialogInterface.OnClickListener(){
            public void onClick(
                DialogInterface dialoginterface, int i){
```

```
                }
        })
```

"setPositiveButton"、"setNegativeButton" 或 "setNeutralButton" 函式都可以用来定义按钮，各按钮分别预设代表正面/中立/负面的结果。

上方程序码中定义的"setPositiveButton"里，包含了一个没有作用的对话框界面(DialogInterface)。 表示当我们按下按钮时，不做任何事就直接退出对话框。

完整对话框函式的程序码如下

代码：

```
private void openOptionsDialog() {
                new AlertDialog.Builder(Bmi.this)
                .setTitle(R.string.about_title)
                .setMessage(R.string.about_msg)
                .setPositiveButton(R.string.ok_label,
                                new DialogInterface.OnClickListener(){
                                        public void onClick(
                                                        DialogInterface
dialoginterface, int i){
                                        }
                                })
                .show();
        }
}
```

更详细的对话框使用可参考官方文件 http://code.google.com/android/referenc ... ilder.html

顺道一提 -"Toast "界面元件

对话框的使用模式，限制了使用者得按下某些按键以跳出对话框，才能继续使用原本程序。如果我们只是要显示一小段提示讯息，而不想打扰使用者的注意力，有没有 更适合的方法哩？ 有的，我们可以把显示方式比较有弹性的对话框拿掉，改为使用简单的 "Toast " 界面元件。"Toast "界面元件的作用是弹出一个讯息框，快速在屏幕上显示一小段讯息。

程序码如下：

代码：

```
import android.widget.Toast;
...
private void openOptionsDialog() {
    Toast.makeText(Bmi.this, "BMI 计算器", Toast.LENGTH_SHORT).show();
      /*new AlertDialog.Builder(this) //注解掉原本的对话框
      ...
      */
    }
```

打开模拟器。我们按下"计算 BMI 值"按钮后，屏幕上不再出现一个对话框，而改成弹出一段"BMI 计算器"文字讯息，过几秒之后即自动隐去。

整段程序值得注意的一行是

代码：

Toast.makeText(Bmi.this, "BMI 计算器", Toast.LENGTH_SHORT).show();

我们对 Toast 元件指定了欲显示文字，与 Toast 元件的显示时间长短(LENGTH_SHORT，即短讯息)，最后与处理对话框一样，呼叫 "show() "方法来将 Toast 元件显示在屏幕上。

错误处理

解决出错最好的方式就是阻止它们的发生。

虽然在当前的程序中，我们似乎用不到 Toast 元件。不过，其实我们还是可以巧妙地运用它。

使用者在输入资料时，难免会出错。而现在我们写好的 BMI 程序中，并没有对使用者可能的输入错误做处理。

因此在下面的程序改进中，我们使用了 try...catch 语句，与 Toast 元件来做错误处理。

代码：

```
DecimalFormat nf = new DecimalFormat("0.00");
        try{
                double height = Double.parseDouble(field_height.getText().toString())/100;
                double weight = Double.parseDouble(field_weight.getText().toString());
                double BMI = weight / (height * height);
                //Present result
                view_result.setText(getText(R.string.bmi_result) + nf.format(BMI));
                 //Give health advice
                if(BMI>25){
                    view_suggest.setText(R.string.advice_heavy);
                }else if(BMI<20){
                    view_suggest.setText(R.string.advice_light);
                }else{
                    view_suggest.setText(R.string.advice_average);
                }
        }
        catch(Exception err)
        {
            Toast.makeText(Bmi.this, "打错了吗？只能输入数字喔",
Toast.LENGTH_SHORT).show();
        }
```

讲解

```
Try
{
主要程序流程
}
catch(Exception err)
{
错误处理流程
}
```

try...catch 语句是 Java 语言的错误处理语句。首先将"主要的程序流程"包在 try 语句的两个大括号中执行，如果执行正常，就跳过 catch 语句，继续执行后续的语句。旦是如果在 try 语句中执行的流程出现错误了，那麼程序流程就会从 try 语句跳到对应的 catch 语句，并开始执行对应的 catch 语句大括号中的"错误处理流程"。

在我们的 BMI 程序中，catch 语句大括号中的"错误处理流程"，是简单地在屏幕上显示一串提示使用者输入错了，应该输入数字的讯息。

代码：

```
Toast.makeText(Bmi.this, "打错了吗？只能输入数字喔", Toast.LENGTH_SHORT).show();
```

需要在屏幕上显示一串提示时，就是 Toast 元件派上用场的地方。除了所显示的字串不同之外，整段程序与上一节相同。

重构

为了更好地重用，我们继续把字串提取到"res/values/strings.xml"中

代码：

```
....
<string name="input_error">打错了吗？只能输入数字喔</string>
</resources>
```

然后在程序中使用"R.string.input_error"来取得字串

代码：

```
Toast.makeText(Bmi.this, R.string.input_error, Toast.LENGTH_SHORT).show();
```

初识 Inten

初识 Intent

开启网页

我们已经对 Android 应用程序的写法有了概观的认识。可是我们还没有触及 Android 平台的特别之处：整合网络的应用。

在上一章中，我们学到如何使用对话框，与如何在对话框下添加按钮，以回到原画面。在这一章里，我们来为我们的应用程序添加一个简单的网路功能：在上一章实做的"openOptionsDialog"对话框函数中，新添一个"连线到首页"的按钮。

我们先把"openOptionsDialog"函数中使用到的字串增加到"res/values/string.xml"里。因此完整的"res/values/string.xml"档桉如下：

代码：

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <resources>
3      <string name="app_name">BMI</string>
4      <string name="height">身高 (cm)</string>
5      <string name="weight">体重 (kg)</string>
6      <string name="bmi_btn">计算 BMI 值</string>
7      <string name="bmi_result">你的 BMI 值是 </string>
8
9      <string name="about_title">关于 Android BMI</string>
10     <string name="about_msg">Android BMI Calc 0.6\n
11         作者 gasolin\n
12         gasolin+android [at] gmail.com</string>
13     <string name="ok_label">确认</string>
14     <string name="homepage_label">首页</string>
15 </resources>
```

增加了"连线到首页"按钮，完整"openOptionsDialog"函数的新版程序码如下：

代码：

```
1   private void openOptionsDialog() {
2       new AlertDialog.Builder(this)
3           .setTitle(R.string.about_title)
4           .setMessage(R.string.about_msg)
5           .setPositiveButton(R.string.ok_label,
6               new DialogInterface.OnClickListener(){
7                   public void onClick(
8                       DialogInterface dialoginterface, int i){
9                   }
10              })
11          .setNegativeButton(R.string.homepage_label,
12              new DialogInterface.OnClickListener(){
```

```
13                      public void onClick(
14                          DialogInterface dialoginterface, int i){
15                          //go to url
16                          Uri uri = Uri.parse("http://androidbmi.googlecode.com/");
17                          Intent intent = new Intent(Intent.ACTION_VIEW, uri);
18                          startActivity(intent);
19                      }
20              })
21          .show();
22  }
```

讲解

在上一章"openOptionsDialog"函数的基础上，我们在函数中添加了一个"setNegativeButton"方法，以提供另一个"NegativeButton"按钮。

代码：

```
.setNegativeButton(R.string.homepage_label,
    new DialogInterface.OnClickListener(){
            public void onClick(
                DialogInterface dialoginterface, int i){
                    .....
                }
    })
```

与上一章我们将 DialogInterface 中的内容空白不同的是，我们为这个按钮添加了连线到特定网址(首页)的"动作"，当使用者按下"首页"按钮后，程序会开启浏览器，并连线到本专桉的首 页"http://androidbmi.googlecode.com/"。

要完成整个连线的"动作"只需要三行程序码：

代码：

```
//go to url  这是注解
Uri uri = Uri.parse("http://androidbmi.googlecode.com/");
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
startActivity(intent);
```

以下是分行详细的讲解：

代码：

```
Uri uri = Uri.parse("http://androidbmi.googlecode.com/");
```

建立一个 Uri 实体，里面包含我们要连到的网址"http://androidbmi.googlecode.com/"。

在我们第一次在程序码中加入"Uri"时叙述时， "Uri"下方会出现红色的线，表示"Uri"可能是个需要由外部导入（import）的函数或类别。 在"Eclispe"开发环境中，我们可以使用"ctrl-shift-O"（Windows）或"cmd-shift-O"（Mac）来自动在程序开头 的地方导入"android.net.Uri"函数库。

代码：

startActivity(intent);

透过"startActivity" 函数，Android 系统即根据收到不同 "意图"(Intent) 的动作和内容，开启对应的新页面或新程序。

在 Android 平台上，各个 Activity 之间的呼叫与交流都要透过"startActivity"一类的函数来互动。"startActivity" 一类的函数中，最重要需传入的内容就是"意图"(Intent) 。因此我们在后面会进一步阐述"Intent"与"Activity"之间的关系。

代码：

```
Intent intent = new Intent(Intent.ACTION_VIEW, uri);
```

在这行中，我们建立一个"意图"(Intent) 实体，并传入这个意图的"动作"与"内容"。Intent 的格式如下：

代码：

```
Intent intent = new Intent(动作, 内容);
```

我们所建立"意图"(Intent) 中，所传入的"动作"是"Intent.ACTION_VIEW"。"Intent.ACTION_VIEW"是 Android 内建的"动作"之一。在 Eclipse 编辑画面中输入"Intent."时，Eclipse 编辑器会弹出可输入的建议动作选单，我们可以透过这个选单，了解可使用的各种 Intent 内建动作。

"Intent.ACTION_VIEW" 这个动作的意义为：依所提供内容的不同，开启对应的程序以检视内容资料。我们可以把这个动作想成在一般桌面系统上，开启一个已知的档桉类型档桉(比如说一 张照片)，操作系统会用你预设的应用程序(比如说某看图软件)开启这个档桉。本例中提供了"Uri"网址类型的内容 给"Intent.ACTION_VIEW"这个动作，所得到的结果，就是开启浏览器并前 往"http://androidbmi.googlecode.com/" 页面。

再做好一点

前面我们看到 Uri 实体的定义方法：

代码：

```
Uri uri = Uri.parse("http://androidbmi.googlecode.com/");
```

看到 Uri.parse() 中，有一个固定的网址，你应该也会想把它抽取出来，丢到 Resource 中统一管理。那麽我们把程序改写，首先把网址抽取出来，放到"res/values/string.xml"档桉中。"res/values /string.xml"档桉更新如下：

代码：

```
...
<string name="homepage_label">首页</string>
<string name="homepage_uri">http://androidbmi.googlecode.com/</string>
```

我们把 Uri.parse() 函数修改，传入资源识别符号.

代码：

```
Uri uri = Uri.parse(R.string.homepage_uri);
```

糟了，Eclipse 上出现了红线，表示在我们的程序码里，有什麽地方弄错了。 因为我们只修改了 Uri.parse 传入的内容，我们可以很确定是我们传入的内容错了。 我们重新输入"Uri."，利用 Eclipse 的自动提示功能，看看 parse 函数到底接受那些类别的参数。 原来，Uri.parse() 函数并不接受资源识别符号型态的输入。这麽一来，我们就得自行根据资源识别符号，来取得资源识别符号所代表的文字叙述内容。 真正能执行的程序码如下：

代码：

```
Uri uri = Uri.parse(getString(R.string.homepage_uri));
```

在程序中，我们可以使用"android.content.Context"类别中的"getString" 函数(或是getText)，来取得资源识别符号对应的文字。

加入菜单

加入菜单(Menu)

在进一步学习 Intent 与 Activity 之前，我们先来完善我们的应用程序。 在前几章中，我们把 "openOptionsDialog"这个用来弹出对话框的函数，放进"calcBMI"这个按钮元件的"OnClickListener"方法中。现 在，我们要把"openOptionsDialog" 移出"OnClickListener"方法，改成按下"Menu"键后，跳出一个菜单列(Menu Bar)。当我们点击菜单列中的选项后，才弹出"openOptionsDialog" 的对话框。

完整的程序码如下：

代码：

```
1  protected static final int MENU_ABOUT = Menu.FIRST;
2  protected static final int MENU_Quit = Menu.FIRST+1;
3
4  @Override
5  public boolean onCreateOptionsMenu(Menu menu) {
6      super.onCreateOptionsMenu(menu);
7      menu.add(0, MENU_ABOUT, 0, "关于...");
8      menu.add(0, MENU_Quit, 0, "结束");
9      return true;
10 }
11
12 public boolean onOptionsItemSelected(MenuItem item)
13 {
14     super.onOptionsItemSelected(item);
15     switch(item.getItemId()){
16           case MENU_ABOUT:
17                openOptionsDialog();
18                break;
19         case MENU_Quit:
20                finish();
21               break;
22       }
23       return true;
24 }
```

每个菜单都包含两个部分：

1. 建立菜单
2. 处理选项动作

"onCreateOptionsMenu" 函数即菜单列的主体。在 Android 机器或模拟器上按下硬体的"Menu"(菜单)键，所弹出的菜单列即是靠"onCreateOptionsMenu"函数来定义。当我们在 Activity 中定义了 "onCreateOptionsMenu" 之后，按下"Menu"(菜单)键时，就会弹出相对应的菜单列。

当我们在 Android 应用程序的菜单列上选择了相应的选项后，则是依赖"onOptionsItemSelected"函数，来负责处理菜单列中各选项所个别对应的动作。

在上面的程序里，我们定义了"关于..."与"结束"两个菜单列中的选项。我们分部分讲解如下：

建立菜单

在 "onCreateOptionsMenu"函数中，我们定义了两个菜单列中的选项。 分行讲解如下：

代码：

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
     return true;
    }
```

"onCreateOptionsMenu" 这个函数是菜单列的主体，它是一个"public"(公开)的函数。函数传入一个"Menu"（菜单）型别的"menu"参数。"boolean"则表示 函数的返回值必须为"boolean"型别的值。因此在函数最后，我们提供函数一个返回值"true"。"@Override"表示我们要完全重写掉已定 义在"Activity"类别中的这个函数。

基于与 onCreate 函数一样的原因，因为我们把菜单列原本的动作覆载 (Override) 掉了，因此在撰写我们自己的内容前，加上一句"super.onCreateOptionsMenu(menu)"叙述，用来呼 叫"onCreateOptionsMenu"函数执行预设的动作。

代码：

```
menu.add(0, MENU_ABOUT, 0, "关于...");
menu.add(0, MENU_Quit, 0, "结束");
```

Android 每个页面对应到一个 Activity，每个 Activity 都有一个独立的菜单列。对传入的"menu"参数作处理就能改变菜单列的内容。

我们看到，增加一个菜单列中选项的格式如下：

menu.add(0, 识别符号(identifer), 0, 字串或资源识别符号);

最后一栏"字串或资源识别符号"就是显示在萤幕上的叙述。 而"识别符号"的目的则是作为这个选项的标籤，以供后续处理选项动作时，更容易辨认出所对应的选项。

代码：

```
protected static final int MENU_ABOUT = Menu.FIRST;
protected static final int MENU_Quit = Menu.FIRST+1;
```

我们看到 MENU_ABOUT 识别符号的定义，是一个固定的常数型别（static final int）。"Menu.FIRST"则代表识别菜单开头的数字，当然我们也可以把这"Menu.FIRST"代号直接用任意数字替换，看看程序会发生什麼事。

处理选项动作

在"OptionsItemSelected"函数中，我们分行讲解如下：

代码：

```
public boolean onOptionsItemSelected(MenuItem item)
{
    super.onOptionsItemSelected(item);
    return true;
```

}

"onOptionsItemSelected" 这个函数是处理所有选项的主体，和"onCreateOptionsMenu"函数相同，也是一个"public"(公开)的函 数。"onOptionsItemSelected"函数传入了一个"MenuItem"（选项）型别的"item"参数。"boolean"表示函数的 返回值必须为"boolean"型别的值。因此在函数最后，我们提供函数一个返回 值"true"。"super.onOptionsItemSelected(item);"表示我们要先执行已定义在"Activity"类别中原本 的"onOptionsItemSelected"函数内容，后面再接着执行我们为此函数新定义的动作。

代码:

```
switch(item.getItemId()){
```

我们可以用"item.getItemId()"函数来取得在萤幕上选取的选项所对应的识别符号代码(identifer)。

```
switch(识别符号代码){
        ....
    }
```

在 swith 叙述中，我们根据从"item.getItemId()"函数取得的识别符号代码判断， 根据选到的识别符号代码，作相应处理。

代码:

```
case MENU_ABOUT:
        openOptionsDialog();
        break;
    case
        ....
        break;
```

在"onOptionsItemSelected"函数中收到 "MENU_ABOUT" 识别符号时，我们呼叫"openOptionsDialog" 函数来弹出对话框。

代码:

```
case MENU_Quit:
        finish();
        break;
```

在"onOptionsItemSelected" 函数中收到 "MENU_Quit" 识别符号时，我们呼叫 Android 内建的"finish"函数来关闭这个 Activity。因为我们的"BMI"应用程序只由一个"Bmi"Activity 组成，所以当我们呼叫"finish"函数来关闭"Bmi"这个 Activity，就等于直接关闭了这个"BMI"应用程序。

而事实上， 在 Android 平台上，无论是开发者或是使用者，都不需要自己来关闭 Activity。 因为 Android 虚拟机(Dalvik) 接手了什麼时候 Activity 该启动或关闭的工作。整个 Android Activity 的运作流程，将在后续章节中作讲解。

定义 Android 清单

在初见 Intent 一章中，我们已尝试过使用 "startActivity"函数，传入适当的"Intent"，来呼叫浏览器的 Activity。

到目前为止，我们可以由学习 Android 应用程序的经验中归纳得出：所有 Android 程序的运作流程，都定义在 Activity 中。

Android 系统与其他系统很不一样的地方是：它的应用程序并不直接与底层系统紧密结合，而是跑在 Android 框架中。这意思是设计 Android 应用程序时，我们并不需要关心实际上运作的机器是哪一牌的手机或是哪一种嵌入式系统，或使用哪一种架构(ARM、x86、MIPS)。我们要关心的只有 Android 框架提供了那些功能，好让我们能操作这台设备。具体来说就是我们只要知道这台机器的萤幕大小、有没有键盘，有没有支援 GPS 等等讯息，就知道我们写的应用程序是否能在这台机器上顺畅地运作。Android 框架与底层系统的整合的问题完全可以留给韧体工程师来操心。

在 执行"startActivity"函数时，应用程序并不是直接呼叫另一个 Activity，而是将"Intent"(意图)传进 Android 框架中。 Android 框架会查看 "startActivity" 呼叫所传入的动作与 Intent 内容是否在注册表中，如果符合，就启动对应的服务或 Activity。

Android 系统中的每一个应用程序，在安装的过程里，都得事先在 Android 框架中注册、登记这个应用程序所建立的 Activity，并事先注明会使用到的服务。譬如当我们在 Android 上安装我们撰写的 BMI 应用程序时，BMI 应用程序就会向 Android 框架登记相关资讯：BMI 应用程序将会用到 "Bmi"这个 Activity。

这份讯息存在于每个 Android 应用程序专桉根目录下的"AndroidManifest.xml"档桉中。如果我们在程序里，要用到其他应用程序或服务所提供的功能，也需一併在此列出。

在安装应用程序的时候，Android 框架会根据应用程序提供的这份清单，将资讯注册于 Android 框架的注册表中。

备注:

这麼说其实是不太精确的。Android 应用程序的运作流程，存在于四种载体中：

1. Activity (活动)
2. Broadcast Intent Receiver
3. Service
4. Content Provider

各种载体的相关内容会在后续章节提到时作解说。

预设的 Activity 清单

我们使用 eclipse Android 开发工具打开"BMI/AndroidManifest.xml"档桉。切换到"AndroidManifest.xml"分页标签，查看预设的 "BMI/AndroidManifest.xml"档桉源代码：

代码：

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android=http://schemas.android.com/apk/res/android
3      package="com.demo.android.bmi"
4      android:versionCode="1"
5      android:versionName="1.0">
6      <application android:icon="@drawable/icon">
7          <activity android:name=".Bmi" android:label="@string/app_name">
8              <intent-filter>
```

```
9                   <action android:name="android.intent.action.MAIN" />
10                  <category android:name="android.intent.category.LAUNCHER" />
11              </intent-filter>
12          </activity>
13      </application>
14      <uses-sdk android:minSdkVersion="3" />
15 </manifest>
```

我们分行讲解如下：

代码：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android
    ....>
        ....
    </manifest>
```

"AndroidManifest.xml" 这个档桉也是以 XML 格式描述，每个 Android 应用程序都需要一个"AndroidManifest.xml"档桉，每份"AndroidManifest.xml"档桉的开头都会出现这段叙述。而整 个"AndroidManifest.xml"档桉的叙述，都包含在"manifest"(清单)这个主要标签中。

代码：

```
package="com.demo.android.bmi"
```

"package" 是"manifest"(清单)标签的一个特别属性，范例中的内容可用来标明，这个应用程序的进入点存在于"com.demo.android.bmi"这个名称空间/路径中。

代码：

```
android:versionCode="1"
android:versionName="1.0"
```

"android:versionCode" 和"android:versionName"是应用程序版本号。 这两个属性是可选的(非必要)。"android:versionName"是给使用者看的版本号， 如"1.0"、"2.0"。
"android:versionCode"则是开发者用的内部版本号，一般使用流水号。

代码：

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
..</application>
```

"manifest" 标签中主要包含一个"application"标签(备注 1)。"application"标签里面，定义了所有这个应用程序用到的 Activity、服务等资讯。"application"标签中的"android:icon"属性，定义了这个应用程序将显示在 Android 主画面中的应用程序图示。
"android:icon="@drawable/icon""表示应用程序图示的资源档存在于"res/drawable /icon" 中。图示的大小必须超过 64x64 像素（Pixel）。"application"标签中的"android:label"属性可用来指定应用程序将显示在 Home 主画面上的名称。也就是预设刚开好机时，可以从桌面下方拉出的应用程序列表。

代码：

```
<activity android:name=".Bmi" android:label="@string/app_name">
```

...
</activity>

"application" 标签中所有用到的 Activity ，都要包含在一个个"activity"标签中(备注 2)。Activity 是 Android 应用程序与使用者互动的主要元素，当使用者开启一个应用程序，第一个看到的画面就是一个 Activity。若是一个应用程序中包含多个画面时，会定义多个不同的 Activity，我们也必须在"application"标签中，使用多个"activity"标签，为不同的 Activity 添加描述。如果我们已经在程序码中定义好了 Activity ，却忘了在"AndroidManifest.xml"档桉中加入对应的"activity"标签，那麼在执行中呼叫到这个 Activity 的时候，将无法开启这个 Activity。

"activity"标签的"android:name"属性，指出了这个 Activity 所对应的类别(class)。"activity"标签中的"android:label"属性可用来指定应用程序将显示在 Activity 画面上方的名称。也可以在程序码中透过"setTitle(“名称”)"来动态修改。

因为在上一 层"Manifest"标签属性中已经定义了"package="com.demo.android.bmi"",因此在"activity"标签 的"android:name"属性中，".Bmi"代表着"com.demo.android.bmi.Bmi"的简写。也可以写成"Bmi"，一样是 代表"com.demo.android.bmi.Bmi"这个类别。

代码:

```
<intent-filter>
   <action android:name="android.intent.action.MAIN" />
   <category android:name="android.intent.category.LAUNCHER" />
/intent-filter>
```

"intent- filter"标签定义了这个"activity"的性质。"intent-filter"中主要包含了两个标签："action" 跟"category"标签。"action"标签中的"android:name"属性，其内容"android.intent.action.MAIN"表示：这个 Activity 是此应用程序的进入点（就像程序中常见的 main 主程序），开启这个应用程序时，应先执行这个 Activity。。常见的还有"android.intent.action.EDIT"等标签，会在之后章节用上的时候讲解。"category"标 签中的"android:name"属性，其内容"android.intent.category.LAUNCHER"表示：这个 Activity 将显示在 Launcher 的应用程序列表中。

我们把整个档桉合到一起看，可以总结出这个档桉所传达的讯息： 在"com.demo.android.bmi"路径下的"Bmi.java"这个档桉中，已定义了一个主要的 Activity; 当我们打开 Android 的时候，显示的是位于"res/drawable/icon"的图示。一旦我们按下图示来启动这个应用程序，Android 应用程序框架会去寻找到定义了"android.intent.action.MAIN"内容的 ".Bmi"activity，并呼叫执行。

代码:

```
<uses-sdk android:minSdkVersion="3" />
```

Android SDK 1.1 版之后引入了这条叙述。透过指定这个参数，系统可以依此辨别应用程序是否使用相容的 SDK 版本，好决定能否在这台机器上安装执行。"1"代表 Android SDK 1.0，"2"代表 SDK 1.1，"3"代表 SDK 1.5。这也是一个可选填的选项。但如果我们的应用程序要发佈出去，一些强势的通路如 Google Android Market 已规定所有新发佈的应用程序必须指定"android:minSdkVersion"这个参数。

备注 1

除了"application"标签外，还有"uses-permission"(例如允不允许存取 SMS、能否存取联络簿、相机功能)、"permission"、"instrumentation"等主要标签。相关的内容在后续章节用到时再一併解说。

备注 2

除了"activity"标签外，对应于 Android 应用程序的运作流程，还有"service"、"receiver"、"provider"等主要元件。相关内容会在后续章节提到时作解说。

参考资料

* Android manifest http://developer.android.com/reference/ ... idManifest

* Intent Action http://developer.android.com/reference/ ... ntent.html

加入新的 Activity

直观来看，每个 Activity 通常会负责处理一个屏幕的内容(包含界面、选单、弹出对话框、程序动作等)。当我们需要从一个屏幕画面切换到另一个屏幕画面的时候，就涉及到了 Activity 切换的动作。 我们可以将 Activity 看成 MVC 模式中的 Control。Activity 负责管理 UI（详细的 UI 细节可以由资源档读入），并接受事件触发。

以是否需要与其他 Activity 交换资料来区分，Activity 可以粗分为两种类型："独立的 Activity"与"相依的 Activity"。不同类型的 Activity，其动作也不尽相同：

独立的 Activity

独立的 Activity 是不需要从其他地方取得资料的 Activity。只是单纯的从一个屏幕跳到下个屏幕，不涉及资料的交换。 从一个独立的 Activity 呼叫另一个独立的 Activity 时，我们只要填好 Intent 的内容和动作，使用 startActivity 函式呼叫，即可唤起独立的 Activity。例如前几章中，用作开启特定网页的 Activity。

相依的 Activity

相依的 Activity 是需要与其他 Activity 交换资料的一种 Activity。相依的 Activity 又可再分为单向与双向。从一个屏幕跳到下个屏幕时，携带资料供下一个屏幕（Activity）使用，就是单向相依的 Activity; 要在两个屏幕之间切换，屏幕上的资料会因另一个屏幕的操作而改变的，就是双向相依的 Activity。 与独立的 Activity 比起来，相依的 Activity 变化更加精采。

我们会在后续章节中，对相依的 Activity 做进一步的说明。

独立的 Activity

本章将继续透过改进 BMI 应用程序来讲解 Android 应用程序设计。在这个过程中，我们将使用到独立的 Activity。

这章中所做的改动都是为了介绍独立的 Activity，而不是为了让 BMI 程序变得更完整。因此你不妨先将写好的 BMI 程序先压缩备份到其他目录中，再随着后面的教学继续探索 Android。

本章的目的是介绍独立的 Activity，会用到两个屏幕，因此除了原本的一个 XML 描述档与一个程序码档桉之外，我们还会额外再定义一个 XML 描述档与一个程序码档桉，以支援第二个屏幕的动作。

要完成独立的 Activity 的动作，我们要做几件事：

1. 在程序码中建立新 Activity 类别档桉
2. 在清单中新增 Activity 描述
3. 在原 Activity 类别中加入 startActivity 函式

程序码中建立新的 Activity 类别档桉

首先，使用 Navigator 档桉浏览视窗，切换到"src/com/demo/android/bmi"资料夹。在"bmi"资料夹中，现存有 "Bmi.java"与"R.java"两个档桉。我们准备在此建立一个新的 Activity 类别档桉。

在"bmi"资料夹图示上按右键，选择"New->Class"选项。 Eclipse 会跳出一个"New Java Class"对话框。

在对话框中的"Name"一栏上填入"Report"。"Report"的字头需大写，这是 Java 程序语言的默认规则。

在"Superclass" 一栏右方，按下"Browse..."，Eclipse 会跳出"Superclass Selection"对话框。在对话框中的"Choose a type"栏位中输入"activity"，输入框下方的"Matching items"栏位中，

会显示出所有可能的类别。我们选择"Activity - android.app - ..."这个选项，点击右下方的"ok"按钮，回到上一个对话框。

此时，"Superclass"栏位中将填入"android.app.Activity"讯息。按下对话框右下角的"Finish"键，Eclipse 会在"bmi"资料夹中，产生一个对应的"Report.java"档桉。

刚产生（尚未修改过的）的"Report.java"档桉如下：

代码：

```
1  package com.demo.android.bmi;
2
3  import android.app.Activity;
4
5  public class Report extends Activity {
6
7  }
```

在解读程序流程一章中，我们已讲解过 Android 程序码的基本架构，即 XML 描述档与程序码两个主要组成部分。稍后我们要处理建立新程序码的相关工作，包含定义对应的 XML 描述档，与在程序码中，填入这个 class 的内涵。

相关工作

在"res/layout"中新增一个"report.xml"档桉，并把描述使用者界面 一章中讲解过的 xml 档桉复制一份过来：

代码：

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
3          android:orientation="vertical"
4      android:layout_width="fill_parent"
5          android:layout_height="fill_parent"
6      >
7      <TextView
8              android:layout_width="fill_parent"
9              android:layout_height="wrap_content"
10              android:text="Hello World, Bmi"
11 />
12 </LinearLayout>
```

打开"src/com/demo/android/bmi/Report.java"，把解读程序流程一章中讲解过的预设的程序码复制进来：

代码：

```
1  package com.demo.android.bmi;
2
3  import android.app.Activity;
4  import android.os.Bundle;
5
6  public class Report extends Activity {
7      /** Called when the activity is first created. */
```

```
8       @Override
9       public void onCreate(Bundle savedInstanceState) {
10              super.onCreate(savedInstanceState);
11              setContentView(R.layout.report);
12      }
13 }
```

上面的程序码中，我们将

代码：

```
setContentView(R.layout.main);
```

换成了：

代码：

```
setContentView(R.layout.report);
```

以对应我们新定义的 XML 描述档产生的资源识别符号。

清单中新增 Activity 描述

我们再打开"AndroidManifest.xml"档桉，并切换到"Application"分页。在"Application"分页的左下角，我们可 以看到"Application Nodes"栏位中，列出目前已在"AndroidManifest.xml"档桉中定义的所有"Activity"。现在我们就来将 "Report"这个新的 Activity 加入到"AndroidManifest.xml"档桉中。

点击"Application Nodes"栏位右侧的"Add..."按钮，弹出一个小对话框。选择"Activity"后，按下"ok"回到"Application"分 页。"Application Nodes"栏位中会增加一个"Activity"项目。选择这个"Activity"项目后，在"Application Nodes"栏位右方会出现新的"Attributes for Activity"相关栏位。

我们点选 "Name*"栏位右侧的"Browse..."按钮，开启另一个对话框。新的对话框中我们可以选择在程序中现有定义的 Activity。我们选择"Report - com.demo.android.bmi"后，按"ok"键回到"Application"分页。此时"Name*"栏位的内容变成了"Report"，"Application Nodes"栏位中的名称也更新成"Report(Activity)"了。

activity 标签的内容

我们从"Application"分页切换到"AndroidManifest.xml"分页，查看刚刚的动作实际上作了些什麼事。

我们发现，在原本的 activity 叙述下方，新增了一行名为"Report"的 activity 标签，完整的"AndroidManifest.xml"清单内容如下：

代码：

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android=http://schemas.android.com/apk/res/android
3      package="com.demo.android.bmi">
4    <application android:icon="@drawable/icon">
5        <activity android:name=".bmi" android:label="@string/app_name">
6            <intent-filter>
7              <action android:name="android.intent.action.MAIN" />
8                <category android:name="android.intent.category.LAUNCHER" />
9            </intent-filter>
```

```
10          </activity>
11        <activity android:name="Report"></activity>
12    </application>
13 </manifest>
```

手动新增 activity 标签

Android 提供了多种方式来协助我们定义"AndroidManifest.xml"清单档桉，除了使用对话框选择的方式之外，你也可以在原本的"AndroidManifest.xml"档桉中直接修改源代码，加入如下叙述：

代码：

```
<activity android:name="Report"></activity>
```

来得到相同的效果。

修改页面标题文字

如果希望在打开 Report Activity 页面时，标题栏上的文字将会是"BMI 报告"，而不是预设跟 Activity 名称相同的"Report"，我们可以在清单裡的"report activity"标签中，加入标签（label）属性的描述。步骤如下。

1. 在"res/values/"目录下，新建一个 report.xml 描述档，存放 Report 活动页面用到的字串。档桉内容如下：

代码：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="report_title">BMI 报告</string>
    </resources>
```

2. 有了描述档后，我们可以再修改"AndroidManifest.xml"清单，为 Report activity 添加标签(label)属性：

代码：

```
<activity android:name="Report" android:label="@string/report_title"></activity>
```

原 Activity 类别中加入 startActivity 函式

在准备好相关资源、并在"AndroidManifest.xml"清单添加新 activity 描述后，我们来到实际控制整个程序流程的程序码部份。

在程序码中，我们要把原本按下按钮即开始计算，并显示 Bmi 值到同一屏幕的按钮动作，改成按下按钮后跳转到 "Report" 这个 Activity。

首先修改"BMI"这个 Activity。打开"src/com/demo/android/bmi/Bmi.java"档桉，修改"OnClickListener"函式(定 义按下按钮时做的动作)的内容。按下按钮后从 Bmi Activity 切换（跳转）到 Report Activity 的程序片段如下：

代码：

```
1   private Button.OnClickListener calcBMI = new Button.OnClickListener()
2   {
3       public void onClick(View v)
4       {
5           //Switch to report page
6           Intent intent = new Intent();
```

```
7             intent.setClass(Bmi.this, Report.class);
8             startActivity(intent);
9         }
10 };
```

修改好后，开启 Android 模拟器。当我们按下"计算 BMi 值"按钮后，屏幕会切换到"Hello World, Bmi"页面(已经进入了 Report Activity)。在这个页面上，我们并没有办法可以直接回到前一个页面，因为这两个页面都是独立的 Activity。但我们还是可以透过按下硬体的"Undo"键，使屏幕切换回原本输入身高体重的页面 (即回到 Bmi Activity)。

讲解

代码:

Intent intent = new Intent();

我们建立一个新的"意图"(Intent) 实体。

代码:

intent.setClass(Bmi.this, Report.class);

为这个意图指定来源的 Activity 所在 class，与要前往的 Activity 所在的 class。

代码:

startActivity(intent);

将定义好的 intent 传入"startActivity"函式中。"startActivity"函式会将 intent 传入 Android 框架，Android 框架会根据各应用程序在系统中注册的资料(有没有联想到我们刚刚为 Report Activity 增加的 activity 清单描述?)，找出 Report 这个 Activity，并呼叫它。

搞懂之后，原来呼叫一个独立的 Activity，所需的功夫其实很单纯呀。

传送数据到新的 Activity

传送数据到新 Activity

Intent 是一个动作与内容的集合。Intent 像是一串网址，传送到系统并意图靠其他 Activity 来处理网址中所指定的动作跟内容。

前一章中，我们已学过独立的 Activity。Android 使用 Intent 来完成在屏幕间切换的动作。Intent 包含 Activity 间切换所需的动作、分类、传送资料等讯息，就像是 Activity 之间的宅急便一样。

因此当我们得在 Activity 之间交换资料时，需要先了解 Intent 的用法。

Intent 可以分为两种类型："现成的 Intent"与"自订的 Intent"。使用现成的 Intent 的例子，可以参考"初见 Intent"一章。在 Android 清单中作设定时，我们还可以使用 IntentFilter，来过滤和找寻对应的 Intent。而一般开发者在程序中所自行撰写的 Intent，则是透过自订 Intent 来做很多事情。比如切换 Activity、在其间传递各式的资料。

要完成在 Activity 之间透过 Intent 传送资讯的动作，可以分成"传递资讯"与"接收资讯"两部分。

使用 Intent 传递资讯

上一章的范例中，我们新增了一个 Report Activity 页面，但是还没有为新页面填入实值内容。在本章中我们会完成将 BMI 应用程序从一个页面改写成为两个页面："输入页面"(原本的 Bmi Activity)，与"结果页面"(Report Activity)的应用程序。"输入页面"从界面上取得身高、体重值，透过传送 Intent，将值携带到"结果页面"。"结果页面"从 Intent 中取出其携带的身高、体重值，用这两个参数来产生 BMI 报告结果。

打开 "src/com/demo/android/bmi/Bmi.java"，修改"Button.OnClickListener" 函式：

代码：

```
1   private Button.OnClickListener calcBMI = new Button.OnClickListener()
2    {
3          public void onClick(View v)
4          {
5                  //Switch to report page
6                  Intent intent = new Intent();
7              intent.setClass(Bmi.this, Report.class);
8              Bundle bundle = new Bundle();
9              bundle.putString("KEY_HEIGHT", field_height.getText().toString());
10            bundle.putString("KEY_WEIGHT", field_weight.getText().toString());
11            intent.putExtras(bundle);
12            startActivity(intent);
13       }
14  };
```

讲解

代码：

```
Intent intent = new Intent();
intent.setClass(Bmi.this, Report.class);
...startActivity(intent);
```

是的，如果你真的有学懂上一章的内容，那麼你可能会发现：我们准备讲解的这段程序码主体，与上一章中所提到的程序码其实一模一样。这段程序码的作用是透过 Intent 通知系统(Android 框架)：我们将要从 Bmi Activity 这个页面（输入页面）前往 Report Activity 页面（结果页面）。如果把我们多加的用来附加资料的程序码拿掉，这段程序码即原来独立的 Activity 的程序码。

代码：

```
Bundle bundle = new Bundle();
...
intent.putExtras(bundle);
```

相依的 Activity 与独立的 Activity 不同之处，就在于相依的 Activity 会附带传送额外资讯到新的 Activity。这些额外资讯都是靠着 Intent 物件来携带的。

传送 intent 时，我们可以在其上附加一些讯息，比如说本例中我们从输入界面中取出了的身高、体重值，要将身高、体重值传送给 Report Activity 后作计算。这些附加在 Intent 上的讯息都储存在 Bundle 物件中。透过"intent.putExtras(bundle)"叙述，我们将"bundle" 物件附加在 Intent 上，随着 Intent 送出而送出。

代码：

```
bundle.putString("KEY_HEIGHT", field_height.getText().toString());
bundle.putString("KEY_WEIGHT", field_weight.getText().toString());
```

这段程序是实际用来附加资料的程序码。将使用者输入的身高、体重值，储存到 bundle 物件中。Bundle 其实是一种特别定义的映射（map）型别。"KEY_HEIGHT"、"KEY_WEIGHT"是我们为储存在 bundle 物件中的身高、体重值，所指定的"识别符号"。在这边，我们直接把身高、体重值都储存成字串。因为整个程序都是我们控制，到时候在接收的 Activity 一端，再透过"KEY_HEIGHT"、"KEY_WEIGHT"这两个"识别符号"来取得实际的身高、体重值。读出的值也是字串，等值读出来以后，再去 做型别转换就好了。当然你也可以直接把身高、体重值存成数字。

Bundle 型别额外提供了很多 API。在传送 Intent 时，使用 Bundle 型别的物件来携带资料，相当方便。

使用 Intent 接收资讯

在使用 Intent 接收资讯前，我们先来加上"Report"这个 Activity 的界面。

相关工作

打开"res/values/report.xml"档桉，修改如下：

代码：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="report_title">BMI 报告</string>
        <string name="report_back">前一页</string>
    </resources>
```

打开"res/layout/report.xml"档桉，修改如下：

代码：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
```

```
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        >
         <TextView android:id="@+id/result"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text=""
            />
         <TextView android:id="@+id/suggest"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text=""
            />
        <Button android:id="@+id/report_back"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/report_back"
            />
</LinearLayout>
```

这时打开 Eclipse 开发环境的 layout 检视，或是执行模拟器，我们可以看到一个"前一页"按钮，按钮前其实还有两个没有内容的 TextView 界面元件，下一节中我们将在"Report"这个 Activity 中取得从"Bmi"Activity 传过来的身高体重资料，根据这些资料产生报告资讯。

在 Activity 中解开资讯

用作接收 Intent，透过 Intent 携带的资讯来计算出 BMI 值的"src/com/demo/android/bmi/Report.java"完整程序码如下：

代码：

```
1  package com.demo.android.bmi;
2
3  import java.text.DecimalFormat;
4  import android.app.Activity;
5  import android.os.Bundle;
6  import android.view.View;
7  import android.widget.Button;
8  import android.widget.TextView;
9
10 public class Report extends Activity {
11        /** Called when the activity is first created. */
12 @Override
13        public void onCreate(Bundle savedInstanceState) {
14                super.onCreate(savedInstanceState);
15                setContentView(R.layout.report);
```

```
16                  findViews();
17                  showResults();
18                  setListensers();
19 }
20
21        private Button button_back;
22 private TextView view_result;
23 private TextView view_suggest;
24
25 private void findViews()
26 {
27     button_back = (Button) findViewById(R.id.report_back);
28       view_result = (TextView) findViewById(R.id.result);
29       view_suggest = (TextView) findViewById(R.id.suggest);
30 }
31
32 //Listen for button clicks
33 private void setListensers() {
34        button_back.setOnClickListener(backMain);
35 }
36
37 private Button.OnClickListener backMain = new Button.OnClickListener()
38 {
39     public void onClick(View v)
40     {
41         // Close this Activity
42         Report.this.finish();
43     }
44 };
45
46 private void showResults() {
47     DecimalFormat nf = new DecimalFormat("0.00");
48
49     Bundle bunde = this.getIntent().getExtras();
50      double height = Double.parseDouble(bunde.getString("KEY_HEIGHT"))/100;
51      double weight = Double.parseDouble(bunde.getString("KEY_WEIGHT"));
52     double BMI = weight / (height * height);
53     view_result.setText(getString(R.string.bmi_result) +nf.format(BMI));
54
55     //Give health advice
56     if(BMI>25){
57         view_suggest.setText(R.string.advice_heavy);
58     }else if(BMI<20){
59         view_suggest.setText(R.string.advice_light);
```

```
60      }else{
61          view_suggest.setText(R.string.advice_average);
62      }
63
64  }
65 }
```

讲解

整个程序的架构我们在"完成 BMI 程序"与"重构程序"两章中已经详细说明过了。

代码：

```
Bundle bunde = this.getIntent().getExtras();
```

当我们透过 Intent 传到新的 Activity 后，只要使用 Activity.getIntent() 函数，就可以得到传来的 Intent 物件。然后使用"getExtras"函式，就能取得附加在 Intent 上的 bunde 物件。

代码：

```
            double height = Double.parseDouble(bunde.getString("KEY_HEIGHT"))/100;
            double weight = Double.parseDouble(bunde.getString("KEY_WEIGHT"));
```

在当前的 Activity 取得了 bundle 物件后，我们可以透过指定储存在 bundle 物件中的身高、体重值的识别符号"KEY_HEIGHT"、"KEY_WEIGHT"来取出身高、体重值的资料。由于我们传参数值来时，所使用的是是字串 格式，所以我们在此得做个型别转换，将参数从字串转换成双倍精度浮点数(Double)型别。

代码：

```
private Button.OnClickListener backMain = new Button.OnClickListener()
{
    public void onClick(View v)
    {
      // Close this Activity
      Report.this.finish();
    }
  };
```

当按下 backMain 按钮元件后，结束 Report Activity，显示出原本的 Bmi Activity。

不透过 Bundle 交换资讯使用 Intent 传递资讯时，我们看到可以用"setClass"方法来指定要传送到的 Activity。我们也可以使用类似的"setString"、"setInt"方法来指定要透过 Intent 附带传送的参数。

在使用 Intent 接收资讯时，我们再使用"this.getIntent().getData()"方法就能取到参数了。"getData"方法取到的参数一般是字串型态 的。 若事先已经知道传来参数的型别，还可以用比"getData"方法更精确的"getString"、"getInt"等方法来取得参数值。

不过既然有好用的 Bundle 类别可用，为什么要自己把事情弄得复杂呢？

记录与出错

记录与侦错可以分成"在程序中加上除错讯息"，与"在侦错环境中查看除错讯息"两部分。

在程序中加上除错讯息

程序几乎行行都可以出错。要看程序中的哪一部分可能会出错，实在是门很深的学问。要是没有线索，光靠我们的脑袋来追踪判断，或是靠直觉东试试、西改改，这种作法就跟使用巫毒术扎娃娃一样，直到被扎的人哪天身体疼了，就算巫毒作法有效。这样实在不是一种好的除错方式。

如果是程序码语法格式上的问题，我们可以在编译前，就透过开发工具提供的预先编译警示，得到提醒并及早改正。在我们改正好这些语法格式上的问题后，开发工具才允许我们实际编译应用程序。接着，才能将编译好的应用程序上传至模拟器，再开始进一步的测试。

除 了程序码语法格式上的问题，绝大部分会造成大麻烦的，是隐藏在程序逻辑中的问题。这些问题只有在模拟器甚至在实际机器上运行时才会出现。为了解决这些问 题，我们需要一些协助工具。在 Android 平台上，我们可以透过"Log"函式，来达到自行在程序码中加入一个个自订的"记录点"或"检查点"。并可以透过开发环境中的"LogCat"工具来查看 记录。当程序流程每次运作到"记录点"时，相应的"记录点"就会在开发工具记录中输出一笔侦错用的讯息。开发者透过这份记录，来检查程序执行的过程、使用 到的参数，是否与我们期望的结果符合。并依此来辨别程序码中可能出错的区域，好能对症根治造成问题的程序码。

导入 Log 函式

打开"src/com/demo/android/bmi/Bmi.java"档桉，我们在程序中加入一些除错讯息。一段含有记录点(Log)的程序码片段如下

代码:

```
import android.util.Log
....
public class Bmi extends Activity {
    private static final String TAG = "Bmi";
    ....
    Log.d(TAG, "find Views");
    Log.d(TAG, "set Listensers");
```

讲解

就像许多人在学生时代 k 书时，会在课本上使用不同颜色作记号。用不同颜色的色笔，来代表各段课文不同的重要性或是意义。"Log"函式的作用，就像是色笔一样，协助我们在程序码中"作记号"，这些数位记号，会在稍后就介绍到的"LogCat"工具中显示。

Log 的使用格式如下

Log.代号(标签, 讯息);

代号

依据讯息的类型，我们有五种 Log 讯息形式可以用作记录。

1. Log.v (VERBOSE) 详细讯息
2. Log.d (DEBUG) 除错讯息
3. Log.i (INFO) 通知讯息
4. Log.w (WARN) 警告讯息
5. Log.e (ERROR) 错误讯息

一般较常用的是 Log.d(除错讯息) 、Log.w (警告讯息)，和 Log.e (错误讯息)。范例中多使用 Log.d(除错讯息) 。

标签

代码：

```
private static final String TAG = "Bmi";
....
Log.d(TAG, "find Views");
```

Log.(v,d,i,w,e) 的第一个参数，是一个自定的记录标签。在目前的 BMI 应用程序范例中，我们还看不太出来自定记录标签的意义。但是当程序的功能一扩张的时候（例如像在 AppDemos 范例那样，包含各种不同功能），我们可以为不同的功能，给予不同的纪录标签。

讯息

代码：

```
Log.d(TAG, "find Views");
```

在 Log.(v,d,i,w,e) 的第二个参数中，加入我们想要记录的资讯。

实际应用

在 BMI 应用程序中，我们可以在用来处理输入错误的"try...catch"语句中加入"Log"讯息，好让我们得以从记录资料中，追踪到输入错误的情况。

代码：

```
public class Bmi extends Activity {
    private static final String TAG = "Bmi";
    ....
    catch(Exception err)
    {
        Log.e(TAG, "error: " + err.toString());
        Toast.makeText(Bmi.this, getString(R.string.input_error),
Toast.LENGTH_SHORT).show();
    }
```

讲解

代码：

```
catch(Exception err)
    Log.e(TAG, "error: " + err.toString());
        ....
    }
```

"Log.e.."叙述的意思是：根据"catch"到的例外型别的资讯(Exception err)，将资料印出到记录中。

他的记录标签方式

我们也不是一定得为每个 TAG 事先定义好一个记录标签，我们可以用当前的 Activity 名称来做为记录标签：

代码：

```
Log.e(this.toString(), "error: " + err.toString());
```

延伸运用

在实作错误讯息提示前，我们其实可以使用 Log.e 函式，来先将错误讯息记录起来，等到整个程序大致底定了，再来用 Toast 或 AlertDialog 元件，来实作输入错误提示的功能。

在侦错环境中查看除错讯息

在程序中加上除错讯息后，我们可以使用除错模式 (Debug Mode) 运行模拟器，并透过开发工具来查看除错讯息。

侦错工具的正式名称为 Dalvik Debug Monitor Service (DDMS)。

启动模拟器

使用除错模式 (Debug Mode) 运行模拟器（选单列->Run->Debug History->BMI）。

切换到侦错环境配置

点选开发环境右上角的 "Open Perspective"按钮，选择 "Other..."选项。选择后会弹出一个"Open Perspective"（开启环境配置）对话框。对话框中列出了所有可用的环境配置列表，选择 "Debug"。此时，右上角的环境配置图示列中，会多出一个"Debug"环境配置图示。整个开发工具的界面配置也为之一变。在右上角的环境配置图示列 中，点选"Java"环境配置图示，就会回到我们原来的界面配置。

现在先切换到"Debug"环境配置，可以看到右下角 的"LogCat"视窗。其上有五个醒目的V、D、I、W、E 图示，分别代表着五种 Log 形式(Verbose, Debug, Info, Warn, Error)，还有一个绿色的"+"号，与一个红色的"-"号。

模拟器运行时会产生很多的讯息记录(Log)，一不注意就看到眼花了。这时候，我们自订的记录标签（范例中自订的标签是"Bmi"）就派上了用场，正好可以为 LogCat 加上一个过滤器(Log Filter)，只显示与"Bmi"标签相关的讯息记录。

加入讯息记录过滤器(Log Filter)

在"LogCat" 视窗右侧，按下绿色的"+"号，会弹出一个"Log Filter"视窗。在"Log Filter"视窗的"by Log Tag"栏位中填入"Bmi"，并填入任意的"Filter Name"后，按下"ok"按钮。"LogCat"视窗上会多出一个与我们填入的"Filter Name"相同的标签。裡面的内容，即所有标示为"Bmi"的自订讯息记录。

参考资料

* ddms http://code.google.com/android/reference/ddms.html
* debug http://code.google.com/android/intro/tu ... redit.html
* trace view http://code.google.com/android/reference/traceview.html

活动的生命周期

维护一个 Activity 的生命周期非常重要，因为 Activity 随时会被系统回收掉。

生命周期

作者在初级章节中一直努力地传达给读者：编写 Android 平台的基本应用程序，跟编写桌面应用程序的难度，两者并没什麼不同。甚至因为 Android 平台拥有免费、跨平台的开发工具，使得 Android 平台应用程序的开发更为单纯。

但是请别忘了，Android 平台也是个手机操作系统。撇掉其他功能不谈，手机的特性，就是应该能随时在未完成目前动作的时候，离开正在使用的功能，切换到接电话、接收简讯模式...而且在接完电话回来应用程序时，还希望能看到一样的内容。

现在使用者使用智慧型手机，大多已习惯使用多工(Multi-Task)的操作系统(如 Windows Mobile)，可以在用手机听音乐的同时，也执行其他多个程序。同时执行多个程序有它的明显好处，但是也有它的严重的缺点。每多执行一个应用程序，就会 多耗费一些系统记忆体。而手机裡的记忆体是相当有限的。当同时执行的程序过多，或是关闭的程序没有正确释放掉记忆体，执行系统时就会觉得越来越慢，甚至不 稳定。

为了解决这个问题， Android 引入了一个新的机制 -- 生命周期 (Life Cycle)。

行程

应用程序(一个个 Activity)执行的状态称为行程(process)。在 Android 操作系统中，每个应用程序都是一个行程。Android 系统平台(准确的说是 Dalvik 虚拟机)会维护一个唯一的 Activity 历史记录堆叠，并从旁观察每个应用程序行程。系统平台会依照系统的记忆体状况，与 Activity 的使用状态，来管理记忆体的使用。

Activity 类别除了负责运行程序流程，与操作界面元件之外，最重要的，就是它提供了开发者控制行程生命周期的函式。我们已经相当习惯在 OnCreate (建立行程时的行为)函式中，加入我们对这个 Activity 执行流程的控制。在前面遇到的范例中，我们并不需要对除了 OnCreate 之外的行为做出改变。不过理解行程的生命周期，将为我们继续深入 Android 开发打下基础。

为什麼要了解生命周期

Android 应用程序的生命周期是由 Android 框架进行管理，而不是由应用程序直接控制。

通常，每一个应用程序（入口一般会是一个 Activity 的 onCreate 方法），都会占据一个行程(Process)。当系统记忆体即将不足的时候，会依照优先级自动进行行程(process)的回收。不管是使用者或开发者， 都无法确定的应用程序何时会被回收。

一个 Activity 类别除了 OnCreate 函式之外，还预先定义了 OnPause(暂停行程时的行为)、OnResume(继续行程时的行为)等等的基本行为，当从一个 Activity 切换到另一个

Activity 的时候，原本的 Activity 将经过一连串的状态改变。开发者可以在程序中添加一些各状态相对应的流程，每次 Activity 改变状态时，就会执行相对应的流程。

要让使用者有好的使用经验，Activity 需要在各个周期点上负责保管状态、恢复状态、传送资料等工作。

Activity 的状态

Android 的虚拟机(VM)是使用堆叠 (Stack based) 管理。主要有四种状态：

* Active (活动)
* Paused (暂停)
* Stopped (停止)
* Dead (已回收或未启动)

Active (活动)

"Active"状态是使用者启动应用程序或 Activity 后，Activity 运行中的状态。

在 Android 平台上，同一个时刻只会有一个 Activity 处于活动(Active)或运行(Running)状态。其他的 Activity 都处于未启动(Dead)、停止(Stopped)、或是暂停(Pause)的状态。

Paused (暂停)

"Paused"状态是当 Activity 暂时暗下来，退到背景画面的状态。

当我们使用 Toast、AlertDialog、或是电话来了时，都会让原本运行的 Activity 退到背景画面。新出现的 Toast、AlertDialog 等界面元件盖住了原来的 Activity 画面。Activity 处在"Paused"状态时，使用者无法与原 Activity 互动。

Stopped (停止)

Stopped"状态是有其他 Activity 正在执行，而这个 Activity 已经离开萤幕，不再动作的状态。

透过长按"Home"钮，可以叫出所有处于"Stopped"状态的应用程序列表。

在"Stopped"状态的 Activity，还可以透过"Notification"来唤醒。"Notification"会在后面章节中解说。

Dead (已回收或未启动)

"Dead"状态是 Activity 尚未被启动、已经被手动终止，或已经被系统回收的状态。

要手动终止 Activity，可以在程序中呼叫"finish"函式。我们在加入选单一章中已经提到过了。

如果是被系统回收，可能是因为记忆体不足了，所以系统根据记忆体不足时的回收规则，将处于"Stopped"状态的 Activity 所占用的记忆体回收。

记忆体不足时的行为

记忆体不足时，Dalvak 虚拟机会根据其记忆体回收规则来回收记忆体：

1. 先回收与其他 Activity 或 Service/Intent Receiver 无关的行程(即优先回收独立的 Activity)

2. 再回收处于"Stopped"状态的其他类型 Activity（在背景等待的 Activity）。最久没有使用的 Activity 优先回收（比较官方的说法是 "根据 LRU 演算法..."）. 还不够？回收 Service 行程

4. 快不行啦，关掉可见的 Activity/行程

5. 关闭当前的 Activity

当系统缺记忆体缺到开始"4. 关掉可见的 Activity/行程"时，大概我们换机子的时机也早该到啦！

观察 Activity 运作流程

讲了这麼多虚的，我们可以写一些程序来直观查看 Activity 的运作流程吗？

当然可以。在上一章记录与侦错 (Log)中，我们学到的"Log"工具，正好可以在查看 Activity 的运作流程时派上用场。

打开"src/com/demo/android/bmi/Bmi.java"，在程序中加入一些"Log"记录点：

代码：

```
public class Bmi extends Activity {
    private static final String TAG = "Bmi";
    public void onCreate()
    {
        super.onCreate(...);
        Log.v(TAG,"onCreate");
    }
    public void onStart()
    {
        super.onStart();
        Log.v(TAG,"onStart");
    }
    public void onResume()
    {
        super.onResume();
        Log.v(TAG,"onResume");
    }
    public void onPause()
    {
        super.onPause();
        Log.v(TAG,"onPause");
    }
    public void onStop()
    {
        super.onStop();
        Log.v(TAG,"onStop");
    }
    public void onRestart()
    {
        super.onRestart();
        Log.v(TAG,"onReStart");
    }
    public void onDestroy()
    {
        super.onDestroy();
```

```
        Log.v(TAG,"onDestroy");
    }

}
```

讲解

我们为 Activity 的各个状态加入了"Log"记录讯息。当模拟器运行时，我们可以透过"LogCat"工具来查看 Activity 所处在的状态。

上面的七个状态又可以归纳成三组：

1. 资源分配 (Create/Destroy)

完整的 Activity 生命周期由"Create"状态开始，由"Destroy"状态结束。 建立(Create)时分配资源，销毁(Destroy)时释放资源。

2. 可见与不可见(Start/ReStart/Stop)

当 Activity 运行到"Start"状态时，就可以在萤幕上看到这个 Activity。相反地，当 Activity 运行到"Stop"状态时，这个 Activity 就会从萤幕上消失。

当使用者按下 Back 按钮回到上一个 Activity 时，会先到 Restart 状态，再到一般的 Start 状态。

3. 使用者能否直接存取萤幕（Resume/Pause）

当 有个 Toast、AlertDialog、简讯、电话等讯息乱入时，原来的 Activity 会进入"Pause"状态，暂时放弃直接存取萤幕的能力，被中断到背景去，将前景交给优先级高的事件。当这些优先级高的事件处理完后，Activity 就改进入"Resume"状态，此时又直接存取萤幕。

Activity 运作流程

由实际运行的记录来看，我们可以归纳出所有 Android 应用程序都遵循的动作流程：

一般启动

onCreate -> onStart -> onResume

启动一个 Activity 的基本流程是：分配资源给这个 Activity（Create 状态），然后将 Activity 内容显示到萤幕上（Start 状态）。在一切就绪后，取得萤幕的控制权（Resume 状态），使用者可以开始使用这个程序。

呼叫另一个 Activity

onPause(1) -> onCreate(2) -> onStart(2) - onResume(2) -> onStop(1)

这是个先冻结原本的 Activity，再交出直接存取萤幕能力（Pause 状态）的过程。 直到 Activity 2 完成一般启动流程后，Activity 1 才会被停止。

回原 Activity

onPause(2) -> onRestart(1) -> onStart(1) -> onResume(1) -> onStop(2) -> onDestroy(2)

点 Back 按钮可以回到原本的 Activity。

退出结束

onPause -> onStop -> onDestroy

如果程序中有直接呼叫"finish"函式来关闭 Activity 的话，系统假设我们很确定我们在做什麼，因此会直接跳过先冻结(Freeze)的阶段，暂停(Pause)，停止(Stop)，然后销毁(Destroy)。

回收后再启动

onCreate ->onStart ->onResume

被回收掉的 Activity 一旦又重新被呼叫时，会像一般启动一样再次呼叫 Activity 的 onCreate 函式。

当我们使用"Android"手机一阵子，在手机上已经执行过多个应用程序。只要按下"Back"（返回）键，"Android"就会开启最近一次开启过的 Activity。

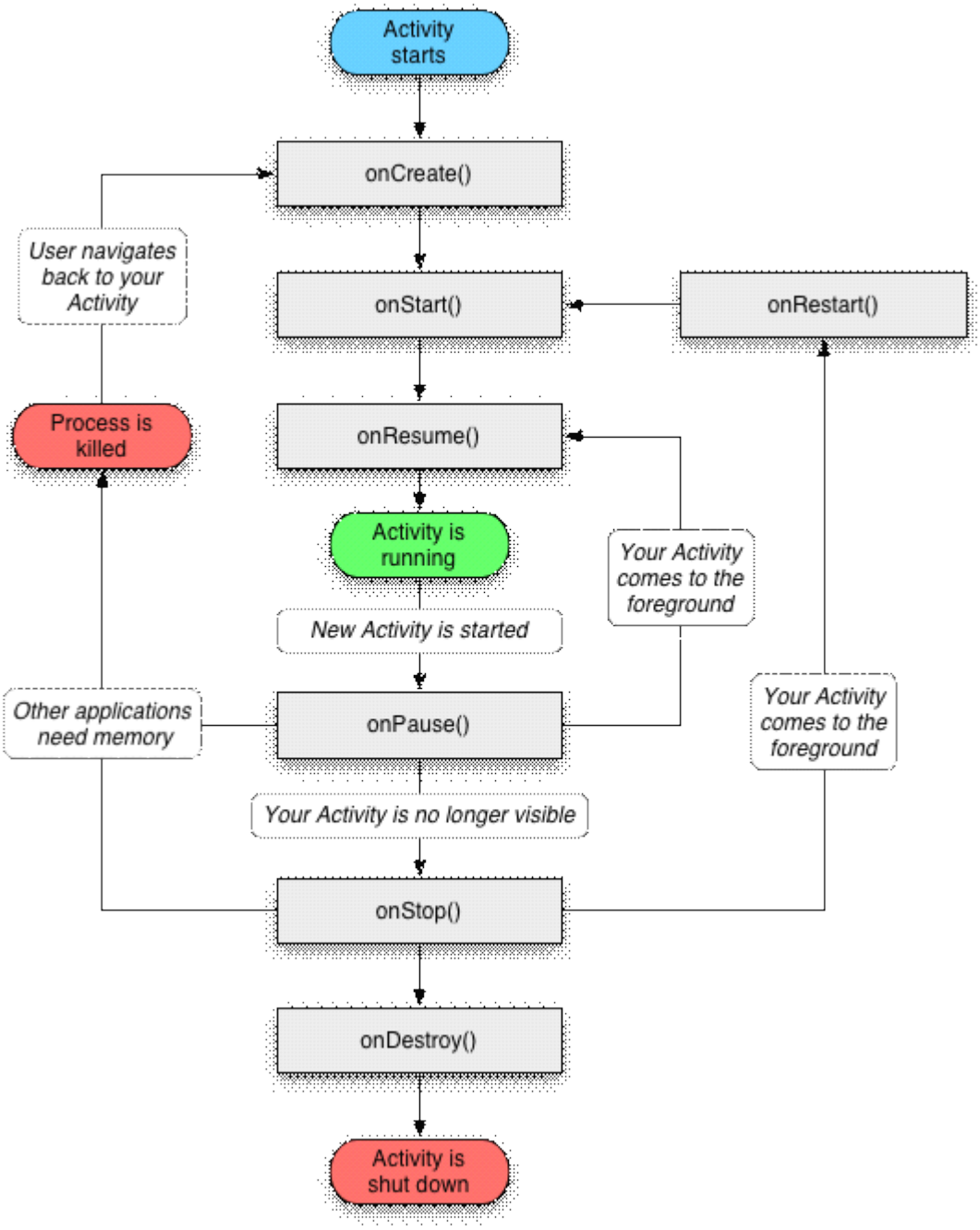
这时我们要是按下多次"Back"（返回）键，理论上迟早会返回到某个已经销毁（Destroy）的 Activity。这时会发生什麼事呢？

如果应该开启的 Activity 已经被回收了，那么这个 Activity 会再次被建立（Create）出来。再次被建立出来的 Activity，当然会跟原本我们开启过的 Activity 不一样啦。

所以如果要让再次被建立出来的 Activity 看起来跟原本开启过的一样，那么在 Activity 之间切换时，我们就要留意保留资料：最好在每次 Activity 运行到"onPause"或"onStop"状态时先保存资料，然后在"onCreate"时将资料读出来。

我们可以使用下章介绍到的"Preference"（偏好设定）等方法来记录之前运作时的资料或设定。

参考资料


Activity starts
onCreate()
User navigates back to your Activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
Your Activity comes to the foreground
New Activity is started
Other applications need memory
onPause()
Your Activity comes to the foreground
Your Activity is no longer visible
onStop()
onDestroy()
Activity is shut down

存储信息

我们都知道，一般人身高的变化程度，比起体重的变化程度小的多。

因此就设计一款 BMI 计算程序来说，如果能在使用者第一次输入身高体重值后，程序能帮我们预先记住上次输入过的身高，那麼等到下次启动程序时，便只需要输入体重。这麼一来，减少了使用者重複输入的麻烦，在使用上就更方便了。使用者应该会喜欢这个便利的功能吧。

使用偏好设定

打开"src/com/demo/android/bmi/Bmi.java"，在"onCreate"和"onStop"中加入"Preference"(偏好设定)相关的程序码。完整的程序码如下：

代码：

```
ublic class Bmi extends Activity {
        private static final String TAG = "Bmi";
       public static final String PREF = "BMI_PREF";
       public static final String PREF_HEIGHT = "BMI_Height";

    /** Called when the activity is first created. */
       @Override
       public void onCreate(Bundle icicle) {
           super.onCreate(icicle);
           setContentView(R.layout.main);
           findViews();
           restorePrefs();
           setListensers();
       }

       // Restore preferences
       private void restorePrefs()
       {
           SharedPreferences settings = getSharedPreferences(PREF, 0);
           String pref_height = settings.getString(PREF_HEIGHT, "");
           if(! "".equals(pref_height))
           {
                   field_height.setText(pref_height);
                   field_weight.requestFocus();
           }
       }

    .....
           @Override
           protected void onStop(){
                   super.onStop();
                   // Save user preferences. use Editor object to make changes.
                   SharedPreferences settings = getSharedPreferences(PREF, 0);
```

```
                settings.edit()
                  .putString(PREF_HEIGHT, field_height.getText().toString())
                  .commit();
        }
```

讲解

取得偏好设定

代码:

```
    // Restore preferences
        private void restorePrefs()
        {
            SharedPreferences settings = getSharedPreferences(PREF, 0);
            String pref_height = settings.getString(PREF_HEIGHT, "");
            if(! "".equals(pref_height))
            {
                    field_height.setText(pref_height);
                    field_weight.requestFocus();
            }
        }
```

我们在"onCreate"函式中，加入一行 "restorePrefs"呼叫。并在"onCreate"函式外，再定义一个"restorePrefs"函式如上。

代码:

```
SharedPreferences settings = getSharedPreferences(PREF, 0);
```

我们宣告了一个偏好设定（SharedPreferences）型别"settings"，并使用"getSharedPreferences"函式，来寻 找系统中有无符合以"BMI_PREF"字串（PREF 参数）作为档名的偏好设定档。如果有符合条件的偏好设定档存在的话，就将这个偏好设定指定使用"settings"作为代号来操作。如果没有的 话，"getSharedPreferences"函式会回传 0 给"settings"。

代码:

```
    String pref_height = settings.getString(PREF_HEIGHT, "");
```

我们可以透过"getXXX"函式，来从偏好设定(SharedPreferences)读取不同型别的内容。例如本例中使用"getString"来读取 文字类型的信息。当"PREF_HEIGHT"偏好设定参数存在时，字串"pref_height"就会得到偏好设定参数的内容。如果不存 在"PREF_HEIGHT"这个偏好设定参数时，字串"pref_height"则会得到一个空字串。

代码:

```
if(! "".equals(pref_height))
        {
                pref_height.setText(pref_height);
            ...
        }
```

当"pref_height"字串存在时，我们将 field_height 栏位内容设定成偏好设定参数中取出的值。

代码：

```
field_weight.requestFocus();
```

同时，因为身高栏位已经预先填好了，使用者只需要再填入体重值即可开始计算自己的 BMI 值。但是当程序一执行，预设的焦点栏位（游标）还是停在"身高"栏位上。因此我们可以在"field_weight"栏位识别符号上，使 用"requestFocus"函式，来手动将焦点栏位改到"体重"栏位上。这样当使用者要输入时，如果之前已经输入过"身高"，那麼程序就会自动帮忙填 好上次输入的身高，并把焦点栏位设置到"体重"栏位上，使用者只需直接输入体重数字就可以了。

如果只加入了"取得偏好设定"这段的程序 码，就运行模拟器来看看结果，会发现我们写在"restorePrefs"函式中的程序码，目前都还没有发生作用。这是因为我们尚未在程序中储存任何偏好 设定。接着，我们将在程序中加入储存偏好设定的程序码，好能在开启 Activity 时读到偏好设定。

储存偏好设定

代码：

```
    @Override
    protected void onStop(){
            super.onStop();
            // Save user preferences. use Editor object to make changes.
            SharedPreferences settings = getSharedPreferences(PREF, 0);
            settings.edit()
              .putString(PREF_HEIGHT, field_height.getText().toString())
              .commit();
    }
```

当我们使用"Home"、"Back"按钮或其他方式离开当前的 Activity 时，我们才把身高的值储存到偏好设定中。根据上一章活动的生命週期，我们知道离开当前萤幕的最后一个状态是"Stop"状态。因此我们覆载 (Override)了"onStop"函式，在其中加入储存身高偏好设定的程序码。"super.onStop"的作用是先将原本的"onStop"函 式执行一遍。

代码：

```
SharedPreferences settings = getSharedPreferences(PREF, 0);
```

我们宣告了一个偏好设定（SharedPreferences）型别"settings"，并使用"getSharedPreferences"函式，来寻 找系统中有无符合以"BMI_PREF"字串（PREF 参数）作为档名的偏好设定档。如果有符合条件的偏好设定档存在的话，就将这个偏好设定指定使用"settings"作为代号来操作。如果没有的 话，"getSharedPreferences"函式会回传 0 给"settings"。

代码：

```
  settings.edit()
              .putString(PREF_HEIGHT, field_height.getText().toString())
              .commit();
```

在此我们串接了三个 settings 拥有的函式："edit"、"putString"，和"commit"。要改变偏好设定(SharedPreferences)型别的内容，需要透 过"edit"函式来编辑。编辑结束后，要透过"commit"函式来将改变写到系统中。我们可以透过"putXXX"函式来为偏好设定

(SharedPreferences)填入不同型别的内容。例如本例中使用"putString"来写入文字类型的信息（读者也可以试试用 putInt 或 putFloat 函式来直接将身高值储存成整数或浮点数）。

本例中"putString"函式所执行的动作，是透过"field_height"界面元件识别符号来取得身高的字串后，将字串储存到"PREF_HEIGHT"所代表的偏好设定参数中。

发布到市集(Market)

Android Market

要释出程序让所有使用者使用有三种方式：

1. 发布到 Android Market
2. 自己提供程序线上下载
3. 发布到第三方 Android 应用程序下载网站

"Android Market (市集)"是一个"Android"官方(Google)提供的"Android"应用程序下载网站，同时也内建于所有的"Android"手机中。透过 手机上的"Market"程序，使用者可以直接在"Android"手机上浏览"Android Market"网站，查看各种可供使用的应用程序。看到喜欢的程序可以直接下载安装。也可以透过"Android Market"为这些软件打分、或是交换对这些软件的意见。

我们也可以将写好的应用程序放在自己的网站上提供下载，或是透过其他 的"Android"应用程序下载网站发布。但是，还有哪个地方会比官方的"Android Market"更容易吸引使用者造访呢？所以我们将主要介绍如何将应用程序发布到官方"Android Market"上。

Android Market 的运作方式

"Android Market"的运作方式如下

* 开发者可以将自己写好的软件上传到 Android Market 中。

* 开发者透过 Android Market 贩卖软件的 30% 收入，得分给电信商跟电子收费商(如手机月费帐单或 Google Checkout 等)，所以开发者可以拿到应用程序定价的 70%。

* 注册为"Android Market Developer"要收美金 25 元的"入场费"。推测可能是种为了保证"Android Market"上应用程序的质量，也为了促使开发者写一点收费软件，好让电信商有得分成的策略。

注册 Android Market

前往 http://www.android.com/market/ ，画面右上角有一段"Interested in having your application in Android Market?"叙述，按下其下方的"learn more"按钮，即可开始注册成为"Android"开发者。

开发者用的网址是 http://market.android.com/publish

开发者可以透过"Android Market"发布"Android"应用程序。首先，开发者得注册一个 Google 帐号。然后使用(Google Checkout)以信用卡付出 $25 美元的注册费用。 最后得同意"Android Market"的使用授权协议。

注册一个 Google 帐号不难，相信大部分读者都已经拥有一个 Google 帐号。

在 申请"Android Market"时要填入加上国码的手机号码。台湾加上国码的手机号码为"+8869xxxxxxxx"。"886"是国码，加上一个"0"之 后，"09xxxxxxxx"是你的手机号码。"+"则是"加上国码的手机号码"表示方式。 接着按下"Google Checkout"图示，如果没有"Google Checkout"的话，也需作先设定。一切完成后在"Google Checkout"中勾选"I agree and I am willing to associate my credit card and account registration above with the Android Market Developer Distribution Agreement."。画面会出现"等待信用卡认证的讯息"，并有 "Google Checkout"的确认函寄到我们设定的电子信箱中。接着想要继续登录开发者网页时，会发现这个网页似乎坏掉了。其实是等待信用卡认证完成，需要一点时 间（一两个钟头），等认证好，完成付款程序后，网页就能再次开启。

开启后会出现"Your Registration to the Android Market is approved!You can now upload and publish software to the Android Market."(已经注册完成)讯息。以后点击"Android Market"网页右上角的按钮时，就会进入开发者面板（Developer Console）页面。

在开发者面板画面的左上角是开发者的昵称。昵称旁边可以选择"Edit profile？"（编辑个人资料）来编辑之前填入的"Android Market Developer"资讯。

上传应用程序到 Android Market

选择右下角的"Upload Application"（上传应用程序）按钮，出现应用程序上传画面。各个栏位的作用都写的很明白，也可以为应用程序自行定价。

"Android Market"上所有的程序可分为"应用程序"与"游戏"两大类。选择好大分类后，其下会出现各自可选的子分类。 在"Upload assets"区块中，点选"Application .apk file"旁的"浏览..."按钮，就可以上传已经签署好金钥的".apk"程序。（本书还未提及怎麼释出签署金钥的应用程序）

直接选择"BMI/bin/"目录中的"BMI.apk"的话，会出现

Market does not accept apks signed with the debug certificate. Create a new certificate that is valid for at least 50 years.

Market requires that the certificate used to sign the apk be valid until at least October 22, 2033. Create a new certificate.

这段警告讯息。意思是说我们要上传的"apk"档用的是"debug"的授权金钥，这样是不能用做发布的，我们得要自行签署金钥才成。

如果改选择透过"AndroidManifest.xml"的"Overview"页眉中"Exporting the unsigned .apk"连结，会出现讯息

The apk is not properly signed.

如果验证成功，该栏位上会直接出现该应用程序图标（icon），与所需的存取权限(permissions)数目。

最后按下左下方的"Publish"按钮，即可将应用程序发布到"Android Market"上。

检视成果 - 查看管理界面

"Android Market"的开发者面板（Developer Console）页面上，列出了开发者当前已发布与未发布的应用程序名称与图标。应用程序名称右侧有明显的星号，表示目前的使用者评价。星号旁边的括号表 示当前已给予评价的人数。星号的右方是该程序的定价。最右侧则是应用程序状态，已发布的应用程序状态是"Published"。还未发布的应用程序状态 是"Saved Draft"。

目前只有透过"Android"手机，才能查看关于应用程序的评论。

自行提供程序线上下载

要自行提供程序线上下载的话，需要指定下载档桉的 MIME 类型。可以在"Apache"网页伺服器的".htaccess"设定中加入:

AddType application/vnd.android.package-archive apk

一行，如此一来使用者在浏览器中点选到".apk"档的连结时，浏览器能自动辨识该档桉为"Android"应用程序类型。

布到第三方 Android 应用程序下载网站

请自行参考"参考资料"中的"其他的 Android 应用程序下载网站"。

针对使用者作设计

针对使用者作设计，有没有意义呢？每个人都有自己的一套道理，不如就用数据来说话吧。

在"Android Market"开放给开发者上传应用程序的第一天（美国时间 10/27），作者即将本书中的两个范例程序"aBMI"(英制)（本章的范 例）、"gBMI"(公制)（基础、中阶的范例）上传到"Android Market"上。考虑到当时使用者(美国)主要集中在使用英制的国家，因此预期"aBMI"应用程序会得到比较好的评价。

果然，在第一 天结束之后，"aBMI"(英制)得到 732 次下载，目前"active installs"(仍安装在机器上)的人数为 452 人（比率 61%）。共有 25 个人平均给予 3 颗星的评价。就一个运作相当简单的应用程序而言，比起其他书籍范例的完成度，3 颗星的评价还是算相当可接受的。

至于"gBMI"(公制)则因为不是针对目标使用者设计，得到 602 次下载，"active installs"的人数为 193 人(比率 32%)。只有 11 个人平均给 2 颗星的评价。

因此可以明显看到，"gBMI"不论是下载的人数、安装后继续使用的比率，或是整体评价都要比"aBMI"差一个档次。当 Android 手机在使用"公制"的国家开卖后，相信比例或评价会再次变化。

我们在设计两个应用程序时，同样需花上差不多的时间，但是却得到有相当明显差别的结果。由此可以看出，手机应用程序需针对使用者特性来设计的重要性。

* Android Market http://www.android.com/market/

* Signing and Publishing Your Applications http://code.google.com/android/devel/sign-publish.html

* http://docs.sun.com/app/docs/doc/820-46 ... _TW&a=view

* http://www.anddev.org/viewtopic.php?p=12252

* http://keytool.sourceforge.net/ 其他的 Android 应用程序集散地

* AndAppStore http://andappstore.com/

* MobiHand OnlyAndroid http://onlyandroid.mobihand.com/

* SlideMe http://www.slideme.org/